НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль

(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки
6.050101 “ Комп’ютерні науки ”

на тему «Розробка програмного агента моніторингу та управління вітроенергетичної установки»

Виконав: студент 4 курсу, групи ТМ-51

Задачин Гліб Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ст. викладач Мірошніченко І.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2017р.

ЗАВДАННЯ

на дипломну роботу студенту

_____Задачину Глібу Сергійовичу_____
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка програмного агента моніторингу та управління вітроенергетичної установки»
керівник роботи Мірошніченко Іван Володимирович ст. викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201 р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи Мова програмування C#, Середовище Visual Studio 2019.16.1, .NET Framework

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати основи розробки програмного агента, проаналізувати роботу вітряної електростанції, розробити програмний агент моніторингу та управління вітроенергетичною установкою, який у реальному часі може аналізувати та розразовувати вітровий потенціал обраної місцевості, проводити моделювання роботи вітрогенератора та порівнювати роботу вітрогенераторів різних типів_____

5. Перелік ілюстративного матеріалу Огляд існуючих рішень, функції системи, формули розрахунку, архітектура системи, приклади роботи програмного модулю

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ” 1 ” грудня 2018р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.18	
2.	Вивчення та аналіз задачі	05.02 – 11.02.19	
3.	Розробка архітектури та загальної структури системи	12.02 – 18.02.19	
4.	Розробка структур окремих підсистем	19.02 – 25.02.19	
5.	Програмна реалізація системи	26.02 – 6.03.19	
6.	Оформлення пояснювальної записки	12.04 – 30.05.19	
7.	Захист програмного продукту	18.05.18	
8.	Передзахист	01.06.19	
9.	Захист		

Студент _____ Задачин Г.С.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Мірошніченко І.В.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Пояснювальна записка містить 49 сторінок, включає 20 рисунків, 1 таблицю, 2 формули та 13 посилань.

Метою дипломної роботи є створення програмного агента управління та моніторингу вітроенергетичної установки. Було створено клієнтський додаток за допомогою мови програмування C# у інтегрованій середі розробки Microsoft Visual Studio. Додаток працює з базою даних розробленою за допомогою Microsoft Sql Server.

Розроблено програмний агент моніторингу вітроенергетичної установки з функціями моделювання роботи вітряка на ВЕС, вибору місця розташування ВЕС, порівняння результатів роботи вітряків різного типу, засоби збереження розрахованої інформації у файл.

Ключові слова: програмний агент, ВЕС, вітроелектростанція, моделювання, управління, моніторинг, C#, WPF

ABSTRACT

The explanatory note contains 49 pages, including 20 illustrations, 1 table, 2 formulas and 13 references.

The purpose of the thesis is to create a software agent for the management and monitoring of the wind power station. A client application was created using the C# programming language in the Microsoft Visual Studio IDE. The application works with a database developed by Microsoft Sql Server.

The software agent for monitoring the wind power installation with the functions of modeling the wind power plant, selecting the location of the wind farm, comparing the results of various types of wind turbines, means of saving the calculated information into a file has been developed.

Keywords: software agent, WPP, wind power plant, modeling, management, monitoring, C #, WPF

АННОТАЦИЯ

Пояснительная записка содержит 49 страниц, включает 20 рисунков, 1 таблицу, 2 формулы и 13 ссылок.

Целью дипломной работы является создание программного агента управления и мониторинга ветроэнергетической установки. Было создано клиентское приложение с помощью языка программирования C# в интегрированной среде разработки Microsoft Visual Studio. Приложение работает с базой данных разработанной с помощью Microsoft Sql Server.

Разработан программный агент мониторинга ветроэнергетической установки с функциями моделирования работы ветряка на ВЭС, выбора места расположения ВЭС, сравнение результатов работы ветряков различного типа, средства сохранения рассчитанной информации в файл.

Ключевые слова: программный агент, ВЭС, ветроэлектростанция, моделирование, управление, мониторинг, C #, WPF

ЗМІСТ

ВСТУП

В наш час все частіше постає питання відновлюваної енергетики, зокрема вітроенергетики, яка спеціалізується на використанні кінетичної енергії вітру. Важливою частиною цієї галузі є аналіз місцевості на якій буде встановлено вітроенергетичну ферму. З розвитком вітроенергетичної галузі зростає вибір та тип вітрогенераторів, тому з’являється необхідність у системі, яка б імітувала роботу вітроенергетичної ферми на конкретній місцевості за певний проміжок часу з певним вітрогенератором.

Така система повинна надавати користувачу вибір місцевості для вітроенергетичної установки та проводити аналіз вітроенергетичного потенціалу для обраного місця, на основі якого формується звіт з прогнозованими значеннями вироблення електроенергії за період часу (рік, місяць, день).

Також система може розраховувати прогноз вироблення електроенергії на існуючій вітроелектростанції та надавати погодинну інформацію про швидкість вітру, стан роботи кожного окремого вітряка та вироблення електроенергії, на основі якого можна проводити роботи з оптимізації електростанції, наприклад, розрахунок додаткової закупівлі вітряків.

На даний момент існують рішення, що надають інформацію про вироблення електроенергії вітряком при введенні середньої швидкості вітру за рік, місяць, тощо. Такі системи не є надійними, тому що не проводиться належна робота з визначення вітроенергетичного потенціалу, та зазвичай дозволяють розраховувати вироблення електроенергії тільки для одного виробника вітряних електростанцій не надаючи користувачу аналогів або подробиць розрахунків, якими б можна було оперувати при визначенні місця будівництва вітряної електростанції.

1. ЗАДАЧА РОЗРОБКИ ПРОГРАМНОГО АГЕНТА МОНІТОРИНГУ ТА УПРАВЛІННЯ ВІТРОЕНЕРГЕТИЧНОЮ УСТАНОВКОЮ

Програмний агент розробляється для управління вітроенергетичною установкою, моніторинга роботи вітрогенераторів, аналізу місцевості для впровадження вітроенергетичної станції, аналізу вітрового потенціалу місцевості, допомоги у виборі вітряка для ВЕС шляхом модулювання роботи вітряка на обраній місцевості. Основними задачами програмного агента є управління роботою вітряної електростанції, виявлення порушень у роботі, повідомлення про помилки у роботі вітряків, прогнозування та розрахунок вироблення енергії вітру за вказаний період часу, надання інформації про роботу кожного окремого вітряка та вітроенергостанції в цілому, допомога в аналізі ефективності вітряків різних типів, моделювання роботи вітряків на обраній місцевості для впровадження до ВЕС, аналіз вітрового потенціалу. Програмний агент повинен мати засоби розрахунку вироблення електроенергії при поточній швидкості вітру, мати функціонал для розрахунку швидкості вітру на висоті вежі вітряка, надавати інформацію про стан вітряка, його номінальну потужність, висоту вежі вітряка, ометаєму площу, номінальну швидкість вітру, вироблювану електроенергію при поточній швидкості вітру. Мають бути присутніми засоби порівняння характеристик двох вітряків, порівняння роботи різних вітряків на одній місцевості та порівняння роботи вітряків одного типу на різних географічних точках для аналізу вітропотенціалу обраної точки, мапа, на якій обирається місце забудови вітроенергетичної ферми або проводиться моніторинг на існуючій, засоби пошуку та вибору існуючої вітроенергетичної ферми за адресою або кліком по мапі, засоби знаходження поточної швидкості вітру на вказаній висоті вежі вітряка та розрахунок енергії за вказаний період часу у погодинному режимі або раз на добу за усередненими показниками швидкості вітру, виведення знайденої інформації на екран у вигляді графіку з можливістю детального огляду роботи вітряка за кожну

обрану дату моніторингу, інформація про сумарну вироблену енергію за обрану дату, інформація про зелений тариф, погодинний звіт з роботи вітряка в якому відображається детальна інформація, засоби порівняння роботи вітряків різного типу, порівняння середнього значення виробленої електроенергії на одній місцевості або у різних місцях.

1.1 Опис вхідної інформації

Електронна мапа, на якій обирається місце для проведення моніторингу електростанції. Обрати місце можна або зробивши клік по мапі, або здійснивши пошук місця за адресою у полі пошуку.

Вибір вітроегенератора для розрахунку електроенергії. Вітряк обирається зі списку, який можна доповнювати натиснувши клавішу “Додати новий вітрогенератор”. Для додання нового вітрогенератора потрібно ввести назву вітряка, номінальну потужність, номінальну швидкість вітру, максимальну та мінімальну швидкості вітру, при яких вітряк може працювати, ометаєму площу або радіус лопастей, висоту вежі та ціну.

Початкова та кінцева дати розрахунку роботи вітрогенератора, які обираються з елементів головного вікна програми.

Режим моніторингу вітрогенератора. Можна обрати моніторинг погодинно або один раз на добу. При цьому погодинний моніторинг дає більш точний результат, тому що дані про швидкість вітру оновлюються для кожної години, коли у випадку моніторингу раз на добу береться усереднена швидкість вітру за день. Вибір режимів моніторингу обумовлено тим, що для дослідження динаміки роботи вітроелектростанції легше оперувати усередненими даними за добу, а для моніторингу роботи вітряка за добу або тиждень потрібні точні результати розрахунків.

1.2 Опис вихідної інформації

Графік з розрахованими показниками електроенергії за вказаний період часу погодинно або раз на добу, загальна електроенергія, яку виробив вітряк за вказаний період часу, функціональні кнопки для дослідження кожної окремої дати з обраного періоду часу коли проводився моніторинг, звіт роботи вітряка з його характеристиками, швидкістю вітра на висоті вежі вітряка, виробленої енергії за годину або добу, та погодинною роботою вітряка.

1.3 Задачі програмного агента

Призначення програмного продукту полягає в:

— Автоматизації процесу розрахунку вітроенергетичного потенціалу обраної місцевості;

— Моделювання роботи вітряка на обраній місцевості для аналізу ефективності вітряка на обраній місцевості;

— Моніторинг роботи ВЕС;

— Контроль введення нового вітряка до системи;

— Функції порівняння характеристик вітряків різних типів для подальшого вибору найбільш ефективнішого;

— Функція порівняння роботи вітряків на основі змодельованих даних за час моніторингу;

2. ОПИС ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ПРИ РОЗВ'ЯЗАННІ ЗАДАЧ УПРАВЛІННЯ ТА МОНІТОРИНГУ ВІТРОЕНЕРГЕТИЧНОЇ УСТАНОВКИ

Програмний агент моніторингу та управління вітроенергетичної установки — технічна система, що забезпечує процес управління вітряною електростанцією, моделювання роботи вітряків, прогнозування роботи вітряної ферми за вказаний період часу з різним інтервалом моніторингу, аналіз роботи вітряків, порівняння різних типів вітряків, порівняння роботи вітряків на основі моніторингу їх роботи та допомога в виборі оптимального місця розташування ВЕС шляхом моніторингу вироблення енергії вітру у різних географічних точках.

Наразі існує багато веб-сайтів, що пропонують конструктори для розрахунку електроенергії виробленої вітряком, такі як: helios-house, alternativenergy, stroyday. Кожний з них має функціонал, що забезпечує розрахунок електроенергії з вибором вітряка для розрахунку за рік або місяць. В онлайн калькуляторі helios-house пропонується обрати точку на мапі для проведення розрахунку, вибрати вітрогенератор та висоту мачти вітрогенератора. На основі введених даних генерується графік середньої виробленої електроенергії за добу в місяць впродовж року. Недоліком такої системи є:

— Неточний аналіз вітропотенціалу, тому що береться середня швидкість вітру за місяць;

— Малий вибір вітряків;

— Неможливо зберегти доданий вітряк;

— Немає доступу до результатів моделювання роботи вітрогенератора за день;

— Неможливо вибрати період, за який потрібно провести аналіз;

— Неможливо зберегти результат моніторингу як звіт;

— Немає функції порівняння двох вітряків;

— Немає функції порівняння роботи вітряків для вибору найбільш

ефективного;

— Відсутня інформація про сумарну вироблену енергію;

— Відсутня інформація про можливі відключення вітрогенераторів через замалу швидкість вітру;

— Відсутня інформація про номінальну потужність, номінальну швидкість вітру, технічні характеристики вітряка;

На веб-сайті alternativenergy для розрахунку електроенергії пропонується ввести середню швидкість вітру за рік. Таке рішення не є точним і може запропонувати лише приблизне значення, тому що не проводиться аналіз вітру.

Така система дозволяє обрати вітряк або з невеликого списку вітряків однієї моделі, або ввести характеристики вітряка, який може не існувати взагалі. Система надає малий функціонал та не може надати гнучкість у роботі, додати вітряк, провести аналіз роботи за обраний період часу, порівняти результати роботи вітряків, надати детальний звіт з результатів роботи.

У перелічених системах немає можливості визначити період, за який потрібно провести розрахунок, немає погодинного звіту з роботи, відсутні характеристики вітряків, неможливо зробити порівняння роботи двох різних вітрогенераторів, відсутня функція проведення аналізу вітряного потенціалу місцевості та містить лише усереднені результати роботи вітрогенератора.

3. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ

Одним із найважливіших завдань при розробці програмних продуктів є вибір таких засобів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання, і дали б змогу отримати результат, який повністю задовольняє користувача.

При створенні програмного продукту було використано засоби реалізації, що зображені на рисунку 3.1.

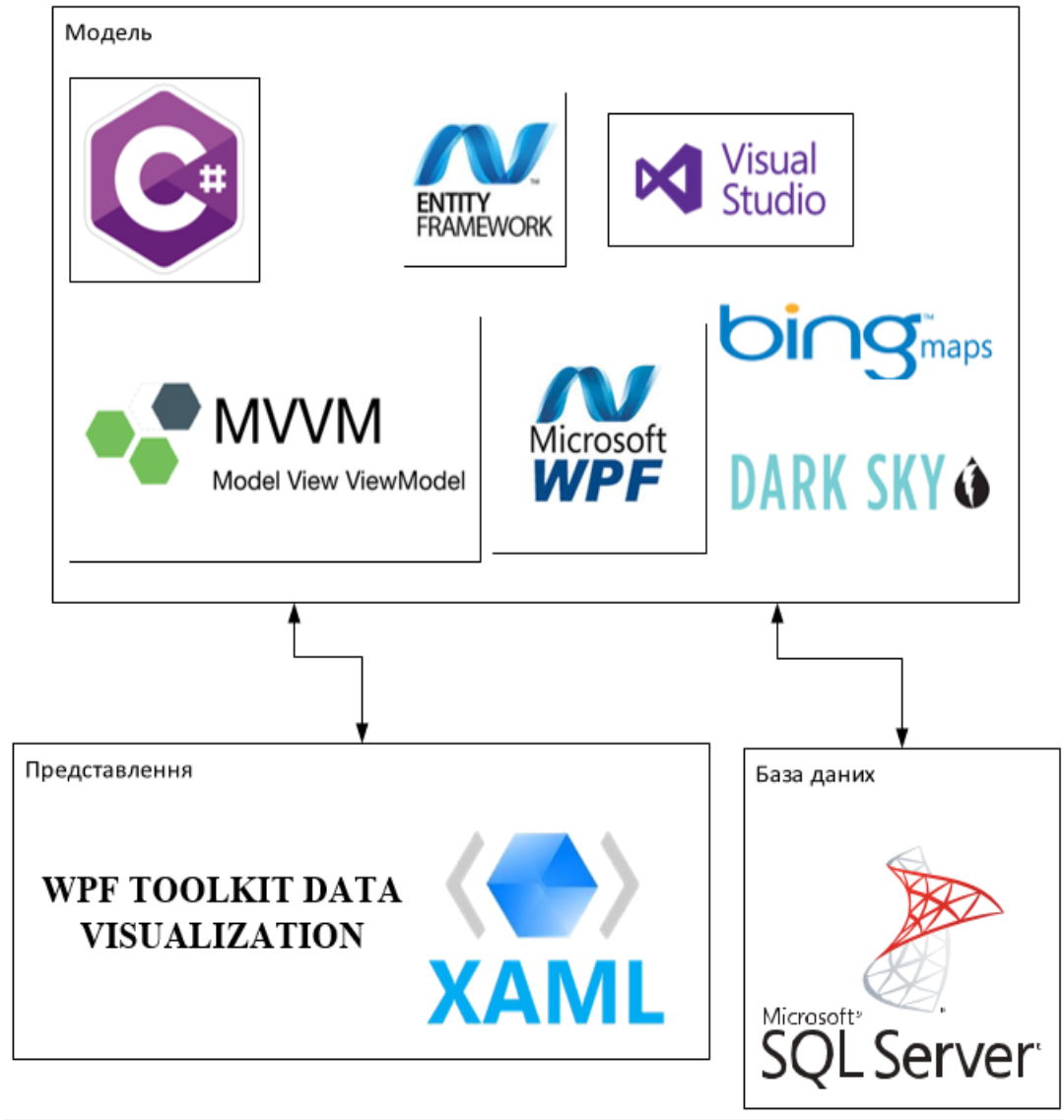


Рисунок 3.1 — Засоби реалізації програмного забезпечення

Як зображено на рисунку 3.1, при створенні програмного забезпечення були

використані такі засоби реалізації:

— Середовище розробки Visual Studio, адже має потужний інструментарій для розробки десктопних програм;

— Мова програмування C#;

— Фреймворк .NET для організації архітектури додатку;

— Шаблон проектування архітектури додатку Model-View-ViewModel для поділу моделі і її представлення, що необхідно для їх зміни окремо один від одного;

— Система для побудови клієнтських додатків Windows Presentation Foundation;

— Технологія Entity Framework для доступу до даних бази даних;

— Прикладний програмний інтерфейс Bing Maps API для додання роботи з мапою до проекту;

— Прикладний програмний інтерфейс DarkSky API для знаходження швидкості вітру за вказаний період часу;

— Git для версіювання розробленої системи;

— Пакет WPF Toolkit Data Visualization для виведення даних у графічному представленні;

— Мова розмітки XAML для організації інтерфейсу;

— База даних Microsoft SQL Server адже вона підтримує розподіленність;

Вибір технологій зумовлений тим, що всі технології мають широку підтримку операційної системи Microsoft Windows та потужний інструментарій для розробки десктопного додатку з можливістю взаємодії з іншими технологіями.

Розробка клієнтського додатку, обумовлена тим, що таке рішення є найбільш простим у використанні та не потребує додаткових навичок у роботі з комп’ютером та не споживає багато оперативної пам’яті. Також така система забезпечує локальну роботу та не залежить від інших компонент.

Мовою програмування високого рівня була обрана C#, адже вона має багато бібліотек, що допомагають спростити процес налаштування застосунку, є частиною .NET, що забезпечує функціонування у середовищі Microsoft Windows, а також

стандартизують архітектуру та стиль написання коду, що важливо для можливого розширення функціоналу іншими розробниками.

Базою даних була обрана MSSQL в першу чергу через швидкодію, адже при накопиченні даних це відіграє важливу роль у продуктивності роботи та є найбільш простою у роботі з .NET Framework.

3.1 Інтегрована середа розробки Microsoft Visual Studio

Розробка програмного агента була проведена за допомогою інтегрованої середи розробки Visual Studio. Microsoft Visual Studio є інтегрованим середовищем розробки (IDE) від Microsoft. Вона використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб і мобільних додатків. Visual Studio використовує платформи розробки програмного забезпечення Microsoft, такі як Windows API, Windows Forms, Windows Presentation Foundation, Магазин Windows і Microsoft Silverlight.

Visual Studio включає в себе редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Інтегрований відладчик працює як відладчик вихідного рівня, так і відладчик на рівні машини. Інші вбудовані інструменти включають в себе профайлер коду, дизайнер форм для створення GUI-додатків, веб-дизайнер, дизайнер класів і дизайнер схеми бази даних. Він приймає плагіни, що підвищують функціональність практично на кожному рівні, включаючи додавання підтримки систем керування версіями (наприклад, Subversion і Git) і додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для доменних мов або наборів програм для інших аспектів розробки.

Visual Studio підтримує 36 різних мов програмування і дозволяє редактору коду і відладчику підтримувати практично будь-яку мову програмування, за умови наявності спеціальної мовної служби.

Найбільш базове видання Visual Studio, видання Community, доступне безкоштовно. Це видання надає повний функціонал для роботи з кодом, підтримує

IntelliSense, роботу з базою даних та систему контроля версій.

3.2 Мова програмування C#

Уся логіка процесів системи написана мовою високого рівня C# — об’єктно-орієнтованою мовою програмування. Вибір зумовлений тим, що платформа C# має велику кількість бібліотек, що були створені для вирішення широкого спектру прикладних задач.

C# — це мова багатопрофільної парадигми програмного забезпечення загального призначення, що охоплює сильні типи, лексично обговорювані, імперативні, декларативні, функціональні, загальні, об'єктно-орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни. C # є однією з мов програмування, призначених для спільної мовної інфраструктури.

За дизайном C # є мовою програмування, яка найбільш безпосередньо відображає базову інфраструктуру спільної мови (CLI). Більшість його внутрішніх типів відповідають типам значень, реалізованим у рамках CLI. Тим не менш, специфікація мови не вказує вимоги до генерації коду компілятора: тобто, не стверджується, що компілятор C# повинен орієнтуватися на Common Language Runtime, або створювати Common Intermediate Language (CIL), або генерувати будь-який інший специфічний формат. Теоретично, компілятор C# може генерувати машинний код, як традиційні компілятори C ++ або Fortran.

Окрім взаємодії з іншими технологіями Microsoft сильною стороною цієї мови є також висока надійність роботи, адже вона розроблялася як об’єктно-орієнтована мова високого рівня з жорсткою типізацію, а самі типи даних залежать від загального середовища CLR, що забезпечує однорідність всій екосистемі .NET. Так, для запобігання неоднозначності та спрощення розуміння коду з мови було виключено множинне наслідування.

У C # покажчики адреси пам'яті можна використовувати лише в блоках, спеціально позначених як небезпечні, а для програм з небезпечним кодом потрібні

відповідні дозволи для запуску. Більшість об'єктів доступу здійснюється через безпечні посилання на об'єкти, які завжди або вказують на "живий" об'єкт або мають чітко визначене нульове значення; неможливо отримати посилання на "мертвий" об'єкт (той, який був зібраний сміттям), або до випадкового блоку пам'яті. Небезпечний покажчик може вказувати на екземпляр типу некерованого значення, який не містить жодних посилань на об'єкти, зібрані зі сміття, масив, рядок або блок пам'яті, виділеної стеком.

Об’єктно-орієнтований підхід до написання коду дозволяє оперувати поняттями, що зустрічаються в реальному житті з певною долею абстракції. Парадигма ООП наділила мову такими властивостями як масштабованість, що дає змогу неодноразово розширювати розроблену систему. Розширюваність системи полягає в тому, що в систему можна додавати нові компоненти, без зміни вже існуючих. Іншою перевагою цього підходу є багаторазове використання написаного коду, що значно скорочує кількість написаного коду. Основу ООП складають чотири основні концепції:

— інкапсуляція;

— успадкування;

— поліморфізм;

— абстракція;

Наразі C# являє собою одну з найрозповсюдженіших мов програмування, адже може застосовуватися для написання багатьох типів програм.

3.3 Фреймворк .NET

Для створення архітектури систем використовують сучасні бібліотеки, чи фреймворки, які спрощують процес написання коду.

Програмний фреймворк — це комплект готових до використання програмних рішень, що включають в себе архітектуру побудови проекту, логіку та базову функціональність системи або підсистеми та, навіть, дизайн.

Фреймворк містить в собі набір бібліотек, що пропонують готовий набір рішень, також може містити допоміжні програми, скрипти та загалом все те, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов’язково об’ємного програмного продукту. Одна з головних переваг використання фреймворків — це стандартизованість структури застосунку.

Фреймоворк .NET — це програма, розроблена корпорацією Майкрософт, яка працює в основному на Microsoft Windows. Вона включає в себе велику бібліотеку класів, названу як Framework Class Library (FCL), і забезпечує взаємодію мови (кожна мова може використовувати код, написаний на інших мовах) на декількох мовах програмування. Програми, написані для .NET Framework, виконуються в програмному середовищі (на відміну від апаратного середовища) з іменем Common Language Runtime (CLR). CLR - це прикладна віртуальна машина, яка надає такі послуги, як безпека, управління пам'яттю та обробка виключень. Таким чином, комп'ютерний код, написаний з використанням .NET Framework, називається "керований код". FCL і CLR разом утворюють .NET Framework.

FCL забезпечує користувальницький інтерфейс, доступ до даних, підключення до баз даних, криптографію, розробку веб-додатків, числові алгоритми та мережеві комунікації. Програмісти розробляють програмне забезпечення, поєднуючи свій вихідний код з .NET Framework та іншими бібліотеками. Фреймворк призначений для використання більшістю нових програм, створених для платформи Windows.

Застосування байт-коду дозволяє отримати крос-платформність на рівні скомпільованого проекту (в термінах .NET: збірка), а не на рівні сирцевого тексту, як, наприклад, в C. Перед запуском збірки в середовищі виконання (CLR) байт-код перетворюється вбудованим в середовище JIT-компілятором (just in time, компіляція на льоту) в машинні коди цільового процесора.

ін'єкція залежностей - це метод, за допомогою якого один об'єкт (або статичний метод) забезпечує залежності інших об'єктів. Залежність - це об'єкт, який можна використовувати (служба). Ін'єкція - це передача залежності на залежний об'єкт

(клієнт), який буде використовувати його. Послуга входить до складу держави клієнта. Передача послуги клієнту, а не дозвіл клієнту побудувати або знайти послугу, є основною вимогою шаблону. .NET Framework має широкий стек технологій (рисунок 3.2).

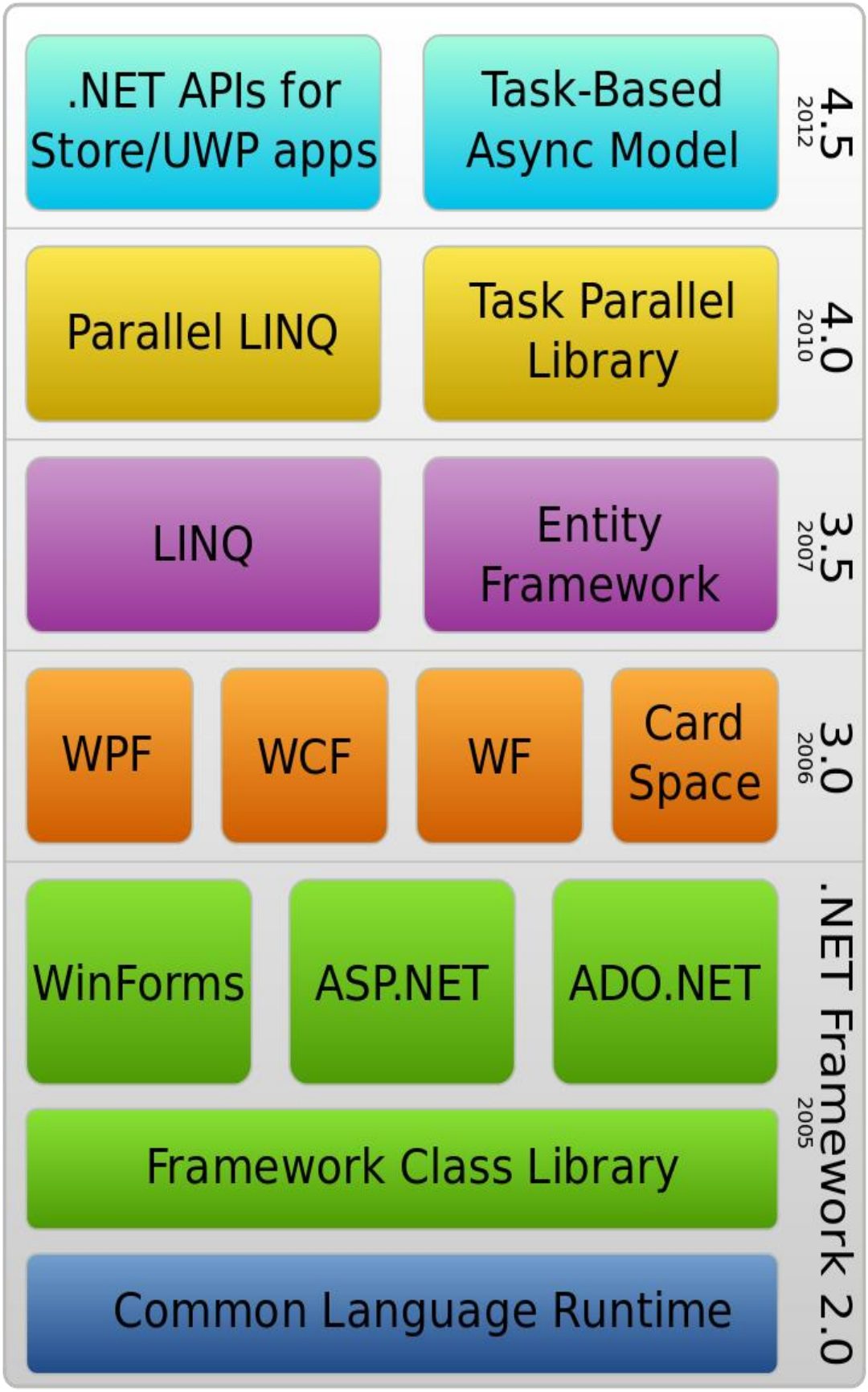


Рисунок 3.2 — Стек технологій .NET Framework

Спільна мовна інфраструктура (CLI) забезпечує нейтральну мовну платформу для розробки та виконання додатків. Реалізуючи основні аспекти .NET Framework в межах CLI, ці функції не будуть прив'язані до однієї мови, але будуть доступні на багатьох мовах, що підтримуються рамками.

Windows Presentation Foundation (WPF) - це графічна підсистема Microsoft для надання користувальницьких інтерфейсів у Windows-додатках. WPF використовує DirectX і намагається забезпечити послідовну модель програмування для побудови додатків. Він відокремлює інтерфейс користувача від бізнес-логіки і нагадує подібні XML-орієнтовані об'єктні моделі, такі як реалізовані в XUL і SVG.

WPF використовує XAML, мова на основі XML, для визначення та зв'язування різних елементів інтерфейсу. Програми WPF можна розгортати як автономні програми на робочому столі або розміщувати їх як вбудований об'єкт на веб-сайті. WPF має на меті уніфікувати ряд загальних елементів інтерфейсу користувача, таких як 2D / 3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка, анімація часу виконання та попередньо відтворені носії. Ці елементи можуть бути пов'язані і маніпулювати на основі різних подій, взаємодій користувача і прив'язки даних.

Бібліотеки часу виконання WPF включені до всіх версій Microsoft Windows після Windows Vista і Windows Server 2008. Користувачі Windows XP SP2 / SP3 і Windows Server 2003 можуть додатково встановити необхідні бібліотеки.

Microsoft Silverlight надала функціональність, яка в основному є частиною WPF для забезпечення вбудованих веб-елементів керування, порівнянних з Adobe Flash. 3D-рендеринг під час виконання Silverlight підтримувався з Silverlight 5.

Після успішної роботи мов розмітки для веб-розробки, WPF впроваджує розширену мову розмітки додатків, яка базується на XML. XAML розроблений як більш ефективний метод розробки інтерфейсів користувача додатків. Конкретна перевага XAML з WPF полягає в тому, що XAML є повністю декларативною мовою, що дозволяє розробнику (або дизайнеру) описувати поведінку та інтеграцію компонентів без використання процедурного програмування. Незважаючи на те, що

рідко додаток будується повністю в XAML, впровадження XAML дозволяє розробникам додатків більш ефективно сприяти циклу розробки додатків. Використання XAML для розробки інтерфейсів користувача також дозволяє розділити модель і вид, що вважається гарним архітектурним принципом. У XAML елементи та атрибути відображаються на класи та властивості базових API.

Як і у веб-розробці, як макети, так і певні теми добре підходять для розмітки, так і XAML не необхідний елемент. Всі елементи WPF можуть кодуватися мовою .NET (C #, VB.NET). Код XAML може бути зібрано в керовану збірку так само, як і всі мови .NET. Одразу після запуску застосунка створюється об'єкт застосунка. У процесі його виконання виникають різні події застосунка, які можна відслідковувати. І, нарешті, коли об'єкт застосунка звільняється, застосунок завершується.

Шаблон Model-View-ViewModel — це шаблон проектування, що застосовується під час проектування архітектури застосунків. MVVM полегшує поділ розробки графічного інтерфейсу користувача - будь то через мову розмітки або код GUI - від розробки бізнес-логіки або локальної логіки (модель даних). Модель перегляду MVVM є перетворювачем величини, що означає, що модель відображення відповідає за викриття (перетворення) об'єктів даних з моделі таким чином, що об'єкти легко керуються і подаються. У цьому відношенні модель перегляду є більш моделлю, ніж відображенням, і обробляє більшість, якщо не всю, логіку відображення перегляду. Модель віображення може реалізовувати шаблон медіатора, організовуючи доступ до локальної логіки навколо набору випадків використання, підтримуваних видом. У таких технологіях, як WPF та Silverlight, присутня концепція “зв'язування даних”, що дозволяє зв'язувати дані із візуальними елементами в обидві сторони. Архітектура MVVM вирішує цю проблему ясним поділом відповідальності:

— Розробка користувацького інтерфейсу здійснюється дизайнером інтерфейсів за допомогою технології, більш-менш природної для такої роботи (XML);

— Логіка користувацького інтерфейсу реалізується розробником як компонент ViewModel;

— Функціональні зв'язки між користувацьким інтерфейсом та ViewModel

реалізуються через біндинги (bindings). Біндинги можуть бути написані в коді або визначені декларативним шляхом;

Шаблон MVVM ділиться на три частини (рисунок 3.3):

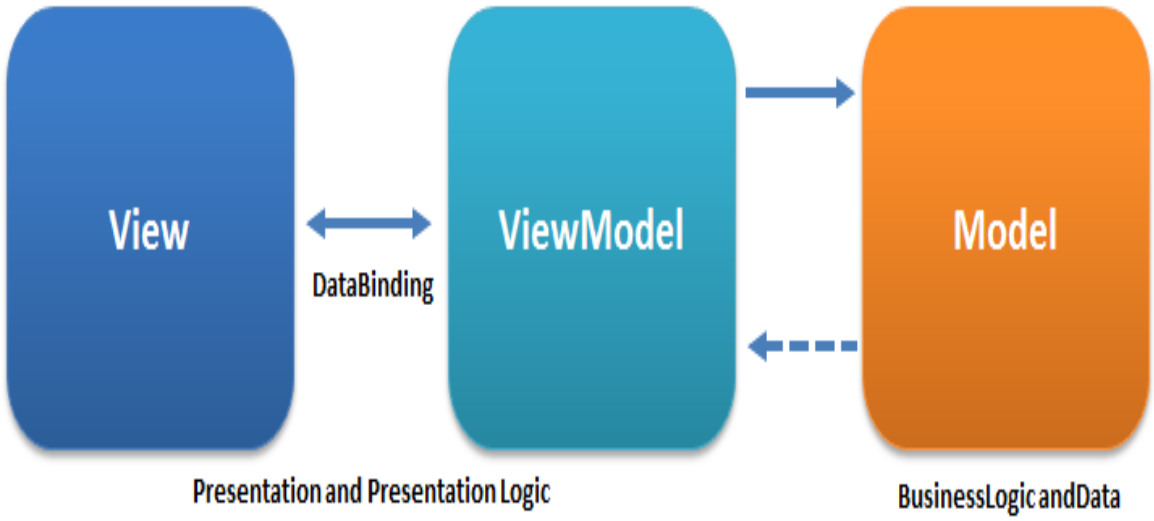


Рисунок 3.3 — Представлення MVVM

— Модель відноситься або до моделі домену, яка представляє вміст реального стану (об'єктно-орієнтований підхід), або до рівня доступу до даних, який представляє зміст (підхід, орієнтований на дані);

— Представлення. Як і в моделях модельного перегляду-контролера (MVC) і моделі-перегляду-презентатора (MVP), представлення - це структура, макет і вигляд того, що користувач бачить на екрані. Він відображає представлення моделі та отримує взаємодію користувача з переглядом (кліки, клавіатура, жести тощо), і він пересилає обробку їх моделі перегляду за допомогою прив'язки даних (властивості, зворотні виклики подій тощо). що визначається для зв'язку моделі перегляду та перегляду;

— Модель перегляду є абстракцією перегляду, що викриває загальні властивості та команди. Замість контролера зразка MVC або презентатора шаблону MVP, MVVM має зв'язуючий пристрій, який автоматизує зв'язок між поданням і його пов'язаними властивостями в моделі перегляду. Модель перегляду була описана як

стан даних у моделі. Основною відмінністю між моделлю перегляду та презентатором у шаблоні MVP є те, що презентатор має посилання на подання, тоді як модель перегляду не має. Замість цього, перегляд безпосередньо прив'язується до властивостей моделі перегляду для надсилання та отримання оновлень. Щоб ефективно функціонувати, для цього потрібна технологія прив'язки або генерування коду шаблону для виконання прив'язки.

— З'єднувач звільняє розробника від зобов'язання писати логіку для синхронізації моделі перегляду і перегляду;

У .NET Framework широко використовується такий шаблон проектування, тому що він надає гнучкий та надійний зв’язок всіх елементів один з одним та є безпосередньо абстрактним рішенням бізнес-моделі.

3.4 Технологія Entity Framework

Entity Framework - це набір технологій в ADO.NET, які підтримують розробку програмно-орієнтованих прикладних програм. Архітектори та розробники програм, орієнтованих на дані, як правило, боролися з необхідністю досягти двох дуже різних цілей. Вони повинні моделювати сутності, відносини та логіку бізнес-завдань, які вони вирішують, і вони також повинні працювати з двигунами даних, що використовуються для зберігання та отримання даних. Дані можуть охоплювати кілька систем зберігання, кожен зі своїми протоколами; навіть програми, які працюють з єдиною системою зберігання, повинні збалансувати вимоги системи зберігання даних до вимог написання ефективного та підтримуваного коду програми.

Технологія Entity Framework дозволяє розробникам працювати з даними у формі предметів і властивостей, специфічних для домену, таких як адреси клієнтів і клієнтів, без необхідності займатися базовими таблицями бази даних і стовпцями, де зберігаються ці дані. За допомогою Entity Framework розробники можуть працювати на більш високому рівні абстракції, коли вони мають справу з даними, і можуть створювати й підтримувати програми, орієнтовані на дані, з меншим кодом, ніж у

традиційних додатках.Технологія Entity Framework є спеціальною об'єктно-орієнтованою технологією на базі фреймворка .NET для роботи з даними. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, то на концептуальному рівні, який нам пропонує Entity Framework, ми вже працює з об'єктами.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами. Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Властивості необов'язково представляють прості дані типу int, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами. При цьому сутності можуть бути пов'язані асоціативної зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Архітектура ADO.NET Entity Framework, знизу вгору, складається з наступного:

— Джерела даних специфічних провайдерів, які абстрактно інтерфейси ADO.NET для підключення до бази даних при програмуванні проти концептуальної схеми;

— Постачальник мап, постачальник конкретних баз даних, який переводить дерево команд SQL команди Entity у запит у власному смаку SQL бази даних. Вона включає в себе специфічний для магазину міст, який є компонентом, відповідальним за переклад загального дерева команд у дерево команд, специфічне для магазину;

— Синтаксичний аналізатор EDM та відображення перегляду, який приймає

специфікацію SDL моделі даних і як вона відображається на базовій реляційній моделі і дозволяє програмувати на основі концептуальної моделі. З реляційної схеми вона створює види даних, що відповідають концептуальній моделі. Вона об'єднує інформацію з декількох таблиць, щоб агрегувати їх у об'єктну сутність, і розбиває оновлення на об'єкт в кілька оновлень до тієї таблиці (таблиць), яка вноситься до цієї сутності;

— Запитувати і оновлювати конвеєр, обробляти запити, фільтри і оновлювати запити на перетворення їх в канонічні дерева команд, які потім перетворюються в запит, специфічних для магазину, постачальником карт;

— Послуги метаданих, які обробляють всі метадані, пов'язані з об'єктами, зв'язками та відображеннями;

— Транзакції, для інтеграції з транзакційними можливостями базового сховища. Якщо базовий магазин не підтримує транзакції, підтримка для нього має бути реалізована на цьому шарі;

— Концептуальний рівень API, час виконання, що виставляє модель програмування для кодування проти концептуальної схеми. Згідно з шаблоном ADO.NET використання об'єктів Connection для звернення до постачальника карт, з використанням об'єктів Command для передачі запиту, і повернення EntityResultSets або EntitySets, що містять результат;

— Від'єднані компоненти, які локально кешують набори даних і набори об'єктних сутностей для використання в рамках оточення ADO.NET Entity Framework;

— Вбудована база даних: ADO.NET Entity Framework містить легку вбудовану базу даних для кешування на стороні клієнта та запит реляційних даних;

— Інструменти проектування, такі як Mapping Designer, також включені в ADO.NET Entity Framework, що спрощує роботу зі збігу концептуальної схеми з реляційною схемою і вказують, які властивості типу сутності відповідають якій таблиці в базі даних;

— Програмування шару, який виставляє EDM як програмні конструкції, які

можуть споживатися мовами програмування;

— Об'єктні служби, автоматично генерують код для класів CLR, які розкривають ті ж самі властивості, що й об'єктна сутність, тим самим дозволяючи реалізацію об'єктів як об'єктів .NET;

— Веб-сервіси, які викривають суб'єкти як веб-служби;

Високорівневі послуги, такі як служби звітності, які працюють на об'єктах, а не реляційні дані.Entity Framework передбачає три можливі способи взаємодії з базою даних (рисунок 3.4):

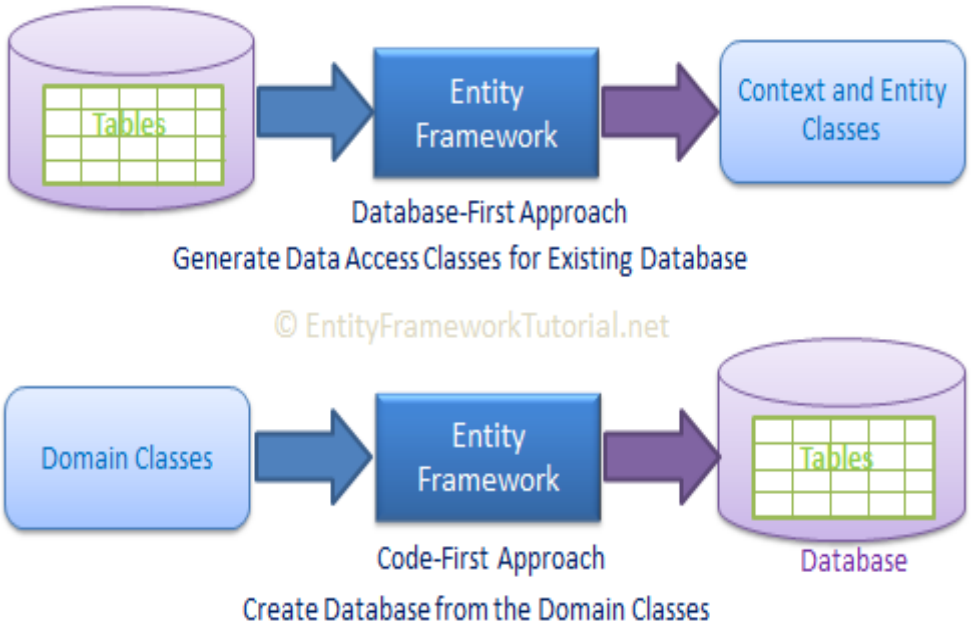


Рисунок 3.4 — Способи взаємодії з базою даних

— Database first: для відображення моделі конкретної бази даних створюється набір класів;

— Model first: розробник спочатку розробляє модель бази даних, по якій потім створюється реальна база даних;

— Code first: розробник створює клас моделі даних, для збереження в базі дпних, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці;

3.5 Прикладний програмний інтерфейс Bing Map API

Платформа Bing Maps надає декілька параметрів API для додатка, включаючи керування Web, керування додатками Windows Store, керування WPF, служби REST і служби просторових даних. Даний програмний продукт використовує технологію Bing Map для визначення локації для моніторингу та передачі інформації про місцевість для отримання швидкості вітру на обраній місцевості.

API — інтерфейс прикладного програмування являє собою набір визначень підпрограм, протоколів зв'язку та інструментів для побудови програмного забезпечення. Загалом, це сукупність чітко визначених методів спілкування між різними компонентами. Хороший API полегшує розробку комп'ютерної програми шляхом надання всіх будівельних блоків, які потім збираються програмістом.

API може використовуватися для веб-системи, операційної системи, системи баз даних, комп'ютерної техніки або бібліотеки програмного забезпечення.

Специфікація API може мати багато форм, але часто включає специфікації для підпрограм, структур даних, класів об'єктів, змінних або віддалених викликів. POSIX, Windows API і ASPI є прикладами різних форм API. Документація для API зазвичай надається для полегшення використання та реалізації. В об'єктно-орієнтованих мовах, API зазвичай означає опис набору визначень класу, з набором форм поведінки, пов'язаних з цими класами. Це поняття пов'язане з реальними функціями, які надані або будуть надаватися, класами, які реалізуються в методах класу.

При розробці додатків API спрощує програмування шляхом абстрагування базової реалізації і лише викриття об'єктів або дій, яких потребує розробник. Хоча графічний інтерфейс для клієнта електронної пошти може надати користувачеві кнопку, яка виконує всі дії для вибірки та виділення нових листів, API для введення / виводу файлів може дати розробнику функцію, яка копіює файл з одного місця в інше без вимагає, щоб розробник розумів операції файлової системи, що відбуваються за лаштунками. У більш загальному плані можна визначити API як сукупність усіх видів

об'єктів, котрі можливо вивести із визначення класу, і пов'язаних з ними можливих варіантів поведінки.

API, зазвичай, пов'язаний із бібліотеками програмного забезпечення: він описує і вказує очікувану поведінку коли бібліотека фактично реалізує вказаний набір правил. Один API може мати декілька реалізацій у вигляді різних бібліотек, які мають той самий інтерфейс.

3.6 Прикладний програмний інтерфейс Dark Sky

Програмний інтерфейс Dark Sky надає широкий функціонал для моніторингу погодних умов. У даній роботі необхідно знати швидкість вітру для вимірювання енергії, що виробляє вітряк, тому для отримання цих даних використовується прикладний програмний інтерфейс від Dark Sky, який отримує на вхід інформацію про місце (широта та довгота) та повертає інформацію про погодні умови на цій місцевості. Dark Sky надає можливість обирати дати, за які потрібно визначити погодні умови. Завдяки цьому програмний агент може змоделювати прогноз роботи вітряка.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Для автоматизації процесу моніторингу та управління вітроенергетичною установкою необхідна система, яка буде надавати найважливішу інформацію та давати можливість контролювати весь процес роботи вітряків з інформацією про стан вітряка та вироблену електроенергію. Система повинна аналізувати швидкість вітру та електроенергію, що генерує вітряк, надавати можливість користувачу порівнювати роботу вітряків різних типів та допомагати у виборі вітряка для встановлення. Кожен вітряк аналізується окремо для надання точної інформації про стан вітрогенератора.

Вибір архітектури клієнтського додатку обумовлений тим, що дана архітектура дозволяє організовувати систему, яка матиме локальне сховище інформації, що необхідно для збереження, додавання та подальшої обробки отриманих даних. Додаток забезпечує постійне функціонування, тому що не залежить від зовнішніх систем та працює автономно. Додаток має функцію додання нового вітряка до системи та вибір початкової та кінцевої дат проведення моніторингу, тож користувач не обмежений у функціях і може повністю контролювати процес роботи системи.

Вибір розподіленої бази даних обумовлений тим, що такий архітектурний підхід забезпечує гнучкість налаштувань кожного окремого вузла (локальної бази даних), збільшує швидкість пошуку даних, адже, в кожній локальній базі зберігаються властиві їй дані та збільшується надійність зберігання даних, адже зменшується вірогідність несанкціонованої зміни даних.

Для організації архітектури програмного продукту було вирішено використати .NET фреймворк та його модулі, адже такий підхід дозволяє розробити гнучкий програмний продукт з чітко виділеними шарами, реалізацію яких можна змінювати без зміни функціональності застосунку.

4.1 Структура програмного забезпечення

Для реалізації задачі управління та моніторингу вітроенергетичної установки був розроблений десктопний застосунок, розроблений за допомогою технології побудови клієнтських додатків WPF. На рисунку 4.1 представлений інтерфейс головного вікна.

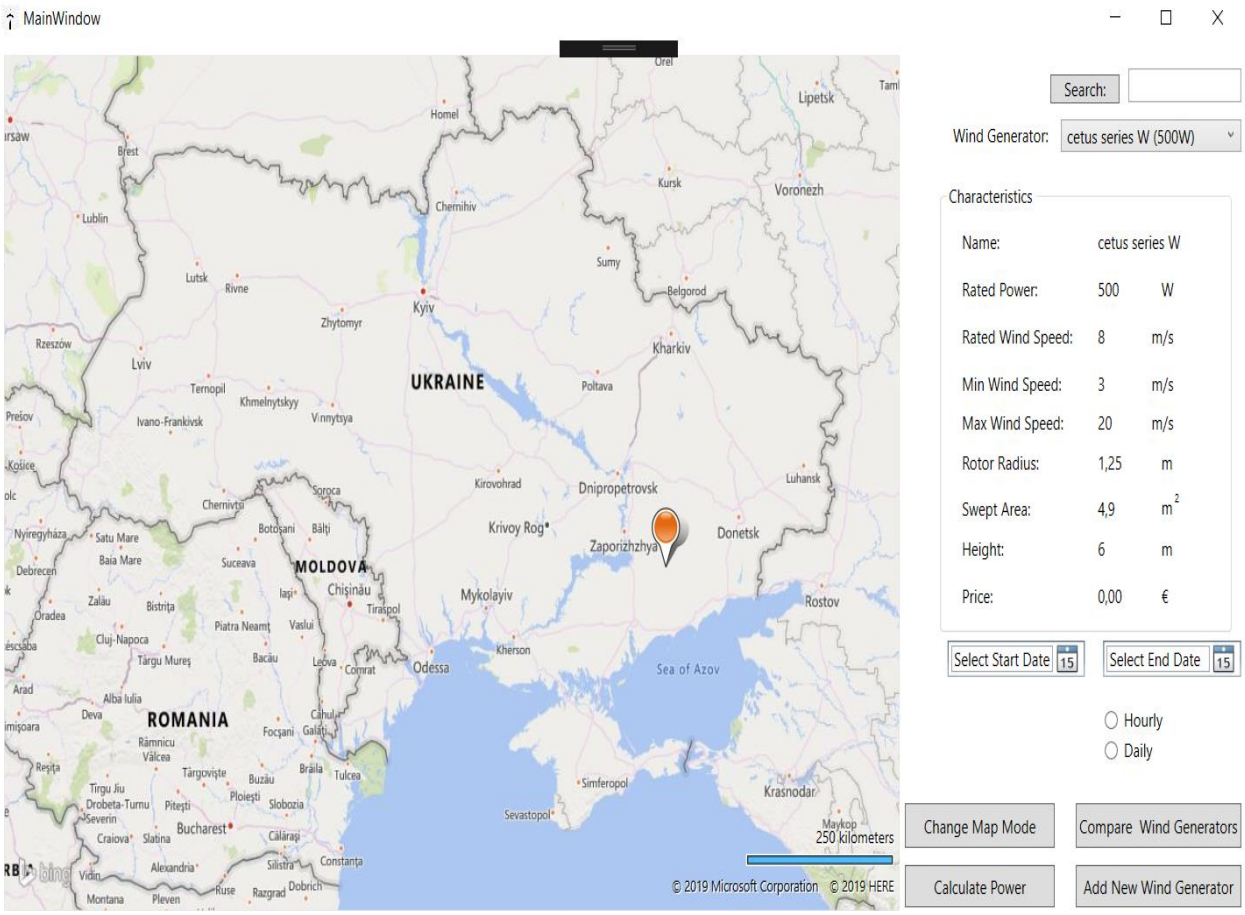


Рисунок 4.1 — Головне вікно додатку

Програмний продукт було реалізовано з використанням .NET фреймворку, це означає, що він має чітко розшаровану архітектуру, де кожен шар може бути замінений без впливу на інші компоненти системи. Структура проекту складається з наступних модулей: Map, Migrations, Weather, WindGenerator, Windows та App.config як показано на рисунку 4.2.

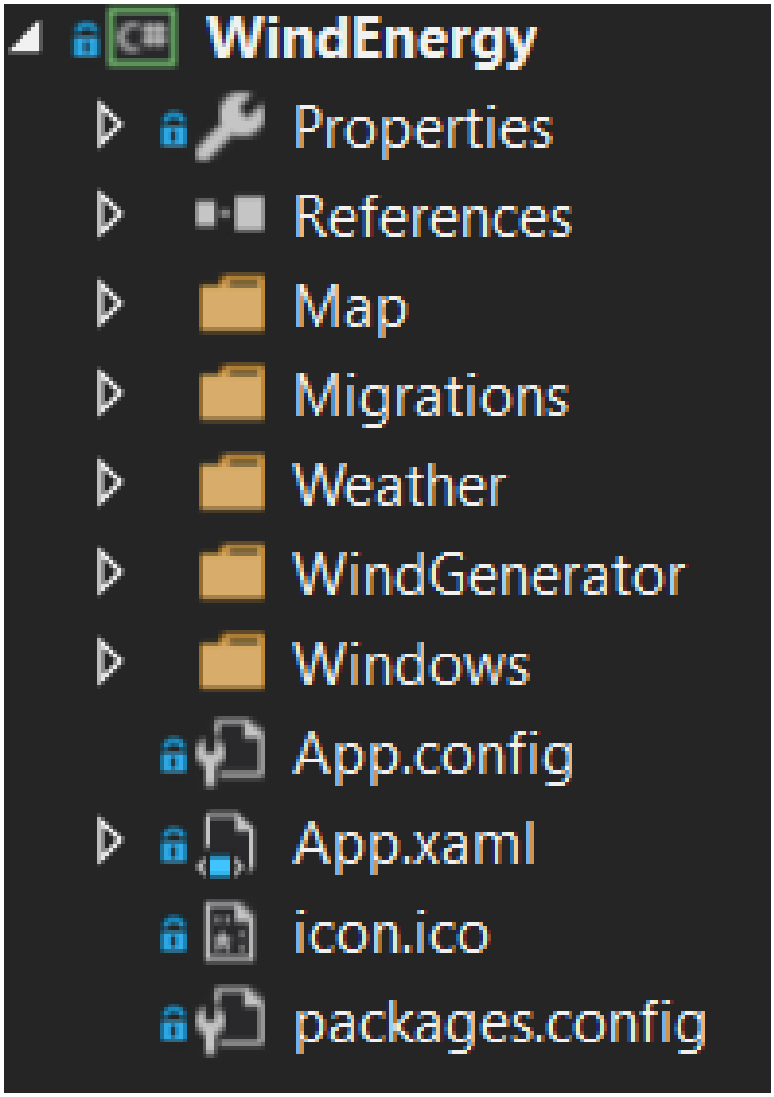


Рисунок 4.2 — Структура файлів проекту

Модуль Map складається з класів, що забезпечують під’єднання та роботу з BingMap API, для роботи якого потрібне підключення до мережі та серіалізатор з XML формату, у якому надходять дані про обране місце моніторингу вітроелектростанції, у клас City, який включає в себе такі параметри як назва місцевості (або адреса), широта та довгота.

Модуль Migrations являє собою набір класів на мові програмування C#, згенерований автоматично Entity Framework при внесенні змін до бази даних. У цьому модулі зберігається інформація про зміни у базі даних та функції оновлення бази даних.

Модуль Weather зберігає необхідні компоненти для підключення до DarkSky API та обробку отриманих даних про швидкість вітру на обраній місцевості за обрану дату.

Модуль WindGenerator включає в себе клас WindGenerator, у якому обчислюється швидкість вітру на висоті вежі вітряка та потужність згенерована роботою вітряка. Також у цьому модулі містяться компоненти для збереження даних у базу даних, створення реляційної моделі класу WindGenerator та допоміжні класи.

Модуль Windows зберігає доступні вікна додатку. Серед них: MainWindow на якому користувач обирає місце, дати моніторингу та тип вітряка, що використовується, AddGenWindow для додання нового вітряка до БД, ChartWindow для виводу графіка роботи вітряка за вказаний час, CompareGeneratorsWindow для порівняння технічних характеристик вітряків, ComparisonWindow для порівняння роботи вітряків за вказаний період часу на одній або різних місцевостях та ProgressBarWindow, який дозволяє відслідковувати прогрес розрахунку виробленої електроенергії вітряком. Кожне вікно містить два елементи з розширенням .xaml та xaml.cs. Перший елемент є графічним відображенням вікна, яке бачить користувач, а другий — це клас C# в якому міститься обробка дій користувача з графічним інтерфейсом.

Модуль App.config — це XML файл, у якому міститься інформація про додаток, модулі, що використовуються, рядок підключення до бази даних та ключі для доступу до API проекту.

4.2 Опис алгоритму розрахунку вироблюваної електроенергії

Функція моніторингу вітроелектростанції є дуже важливою у функціонуванні станції, адже система повинна цілодобово контролюватися та надсилати повідомлення про збої у роботі вітрогенераторів. Також найважливішою функцією системи є аналіз роботи вітряної електростанції, проведення моніторингу виробленої енергії та надання звіту з роботи електростанції за вказаний період.

Для розрахунку виробленої електроенергії потрібно знати швидкість вітру на висоті, що відповідає висоті вітряка. Dark Sky надає швидкість вітру на висоті 10 метрів над землею, тому необхідна формула розрахунку швидкості вітру на вказаній висоті.

Формула 4.1 розраховує швидкість вітру на вказаній висоті

$$V_{\text{шукана}} = V_{\text{відома}} * \left(\frac{H_{\text{шукана}}}{H_{\text{відома}}}\right) \quad (4.1)$$

де

Vшукана — невідома швидкість вітру, м/с

Vвідома — відома швидкість вітру, м/с

Ншукана — висота вітрогенератора, м

Нвідома — висота, на якій вказана відома швидкість вітру, м

Після знаходження швидкості вітру на вказаній висоті потрібно розрахувати електроенергію, що виробляє вітряк за допомогою формули 4.2

$$N = \frac{p*S*V^3}{2} \quad (4.2)$$

де

N — вироблена потужність, Вт

p — густина повітря, $\frac{\text{кг}}{\text{м}^3}$

S — площа, що ометається вітряком, м^2

V — швидкість вітру, м/с

4.3 Опис алгоритму управління та моніторингу вітряної електростанції

Програмний агент управління та моніторингу — це система, що забезпечує процес роботи вітроелектростанції або симуляції роботи для аналізу характеристик обраної місцевості для побудови вітроелектростанції. При цьому передбачається моніторинг електростанції не тільки за поточний день або годину, а також і за обраний користувачем період. Такий функціонал забезпечить процес прогнозування роботи вітрової електростанції у майбутньому та допоможе в оптимізації роботи.

Моніторинг даної системи полягає в тому, що потрібно контролювати процес роботи вітряків, розраховувати поточну електроенергію від кожного вітряка, генерувати інформацію про роботу вітроелектростанції, контролювати процес вимикання та вмикання вітряка коли швидкість вітру досягне максимально допустимої або повернеться до допустимої відповідно.

Для отримання даних про роботу вітряної електростанції система повинна надавати звіт, у якому буде відображена погодинна інформація про технічний стан вітряка, його номінальну потужність, швидкість вітру, вироблену електроенергію при знайденій швидкості вітру.

Отже, у результаті розробки застосунка була розроблена система, що дозволяє:

— Додавати новий вітряк до системи для моніторингу;
— Проводити моніторинг роботи вітряків;
— Надавати інформацію про стан вітряка, вироблену енергію;
— Проводити порівняння роботи вітряків за вказаний період часу;

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

Для роботи системи необхідно встановити додаток та бути підключеним до мережі інтернет для отримання інформації про обране місце моніторингу електростанції та швидкість вітру в обраному місці. При запуску додатку користувач потрапляє на головну сторінку, де він може обрати місце моніторингу для вітряної електростанції на мапі за допомогою подвійного кліку по мапі або здійснивши пошук, увівши адресу у полі для пошуку.

5.1 Системні вимоги

Для роботи користувача з розробленою програмною системою користувачу потрібні мінімальні потужності апаратного забезпечення (вимоги наведені в таблиці 5.1).

Таблиця 5.1. Вимоги до апаратного забезпечення клієнта

Пристрій	Характеристика
Процесор	Intel ® Core ™ 2 / 2 Duo / Pentium ® / Celeron ® / Xeon™ / i3 / i5 / i7 чи AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ з тактовою частотою не нижче 1.5 GHz.
Оперативна пам’ять (RAM − Random Access Memory)	Рекомендовано не менше 1GB RAM
Швидкість з’єднання з інтернет	Рекомендовано не менше 128 кб/сек

Підтримувані апаратні архітектури:

- 32-розрядна (x86);
- 64-розрядна (x64)

5.2 Сценарії роботи користувача в розробленій системі

Перше, що бачить користувач при запуску додатку, це головне вікно застосунку (рисунок 5.1). На цьому вікні користувач може обрати місце на якому бажає провести моніторинг вітряної електростанції одним з двох способів:

— Подвійним кліком по мапі;

— Написати адресу місця у полі пошуку та натисну кнопку “Search” (пошук);

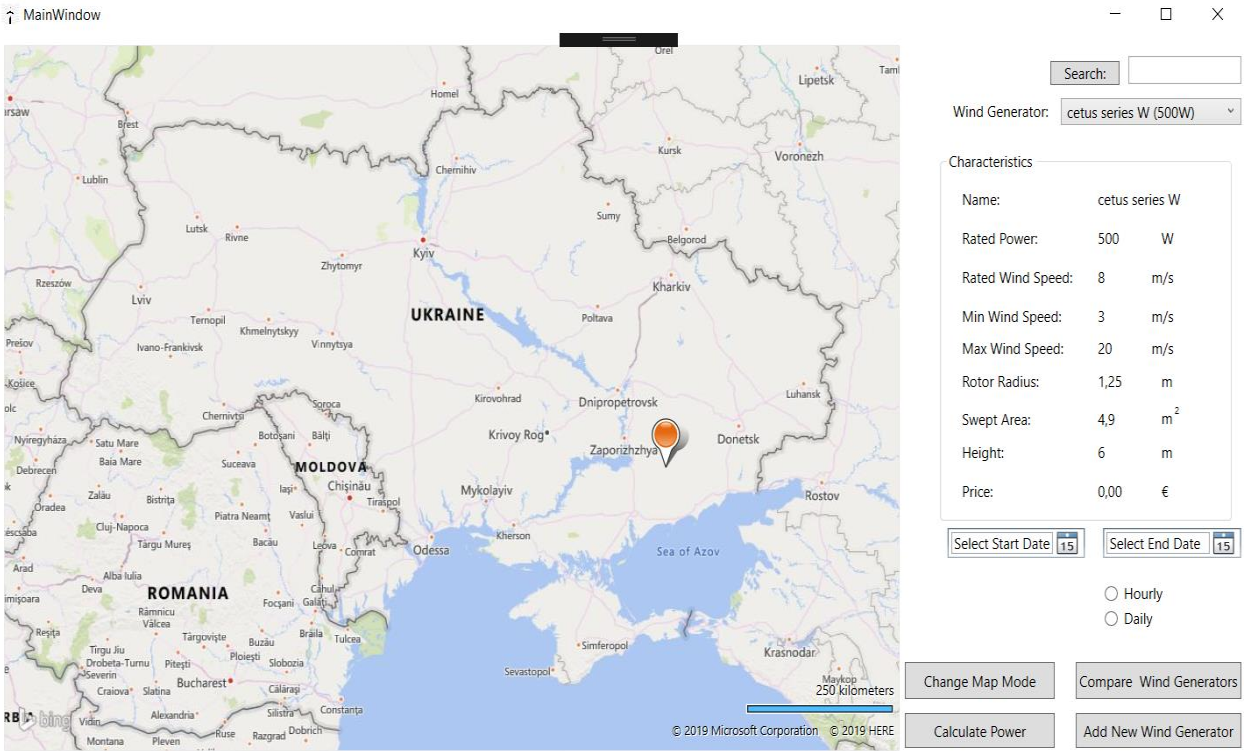


Рисунок 5.1 — Вікно авторизації

Також на головному вікні користувач може обрати один зі списку вітряних генераторів для аналізу та побачити його характеристики. Елементи вікна Start Date та End Date – вбудовані елементи WPF, які містять поля типу DateTime для роботи з часом та датою. Ці елементи потрібні для введення початкової та кінцевої дат проведення моніторингу відповідно. За замовченням обирається моніторинг за поточний день.

Під інформацією про обраний вітряк користувач може обрати частоту проведення моніторингу: один раз на добу або погодинно. Погодинний моніторинг є

найбільш точним тому що аналізує швидкість вітру за кожну годину, коли денний варіант бере усереднену швидкість вітру за добу. За замовченням моніторинг ведеться погодинно.

З головної сторінки користувач може перейти на наступні вікна за допомогою кліка по відповідним кнопкам:

— Вікно моніторинга кліком по кнопці “Calculate Power” (розрахувати потужність);

— Вікно порівняння вітряків кліком по кнопці “Compare Wind Generators” (порівняти вітряні генератори);

— Вікно для додання нового вітряка до бази даних кліком по кнопці “Add new Wind Generator” (додати новий вітряний генератор);

Для зручності у використанні мапи користувач може переключити режим мапи за допомогою кнопки “Change Map Mode” (змінити режим мапи) як показано на рисунку 5.2.

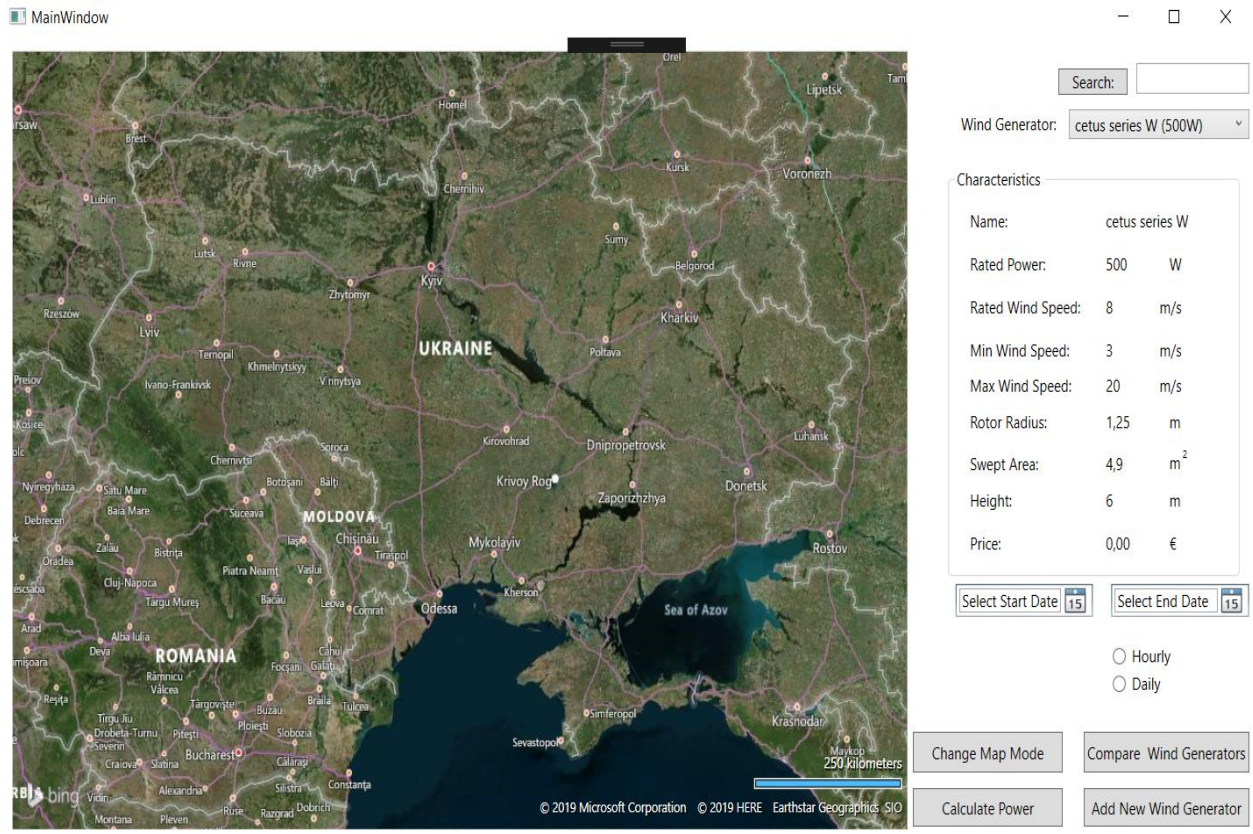


Рисунок 5.2 — Альтернативний режим відображення мапи

При натисненні на кнопку “Додати вітряний генератор” користувач потрапляє на вікно додання вітрогенератора, на якому потрібно ввести наступні характеристики вітрогенератора (рисунок 5.3):

— Назва вітряка;

— Номінальна потужність, Вт;

— Номінальна швидкість вітру, м/с;

— Максимальна швидкість вітру, при якій вітряк може функціонувати, м/с;

— Мінімальна швидкість вітру, при якій вітряк може функціонувати, м/с;

— Радіус ротора (не обов’язково якщо користувач увів ометаєму площу вітряка), м;

— Ометаєма площа вітряка (не обов’язково якщо користувач увів радіус ротора вітряка), м^2;

— Висота вітряка, м;

— Ціна, € (не обов’язково);

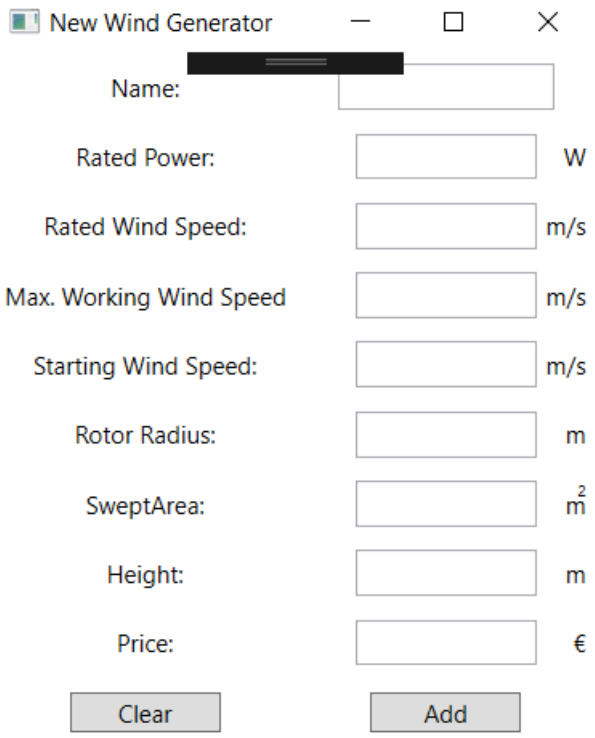


Рисунок 5.3 — Вікно додання нового вітряка

Якщо користувач увів невірні дані, увів назву вітряка, яка вже є у системі або

не увів дані у обов’язкові поля, то побачить повідомлення про помилку як на рисунку 5.4.

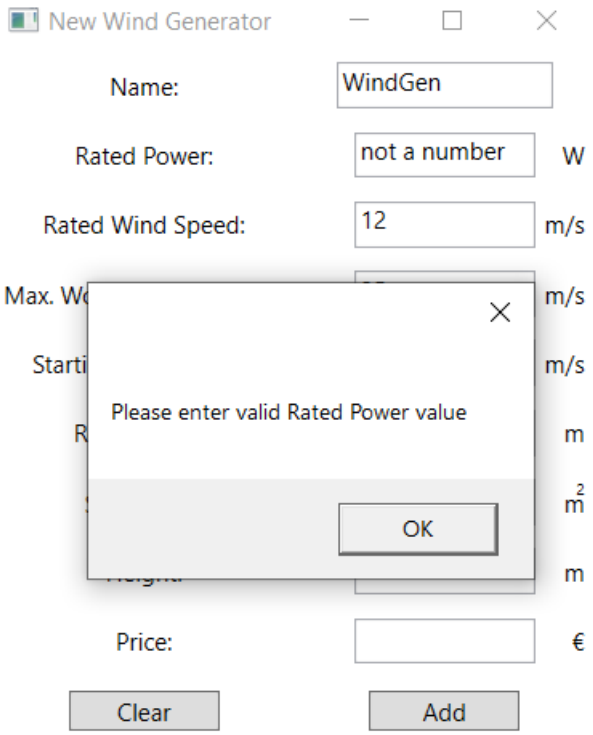


Рисунок 5.4 — Вікно повідомлення про помилку

При натисканні на кнопку “Clear” (очистити) всі текстові поля вікна стануть порожніми.

Якщо користувач увів інформацію про вітряк без помилок та натиснув кнопку “Add” (Додати), то новий вітряк додасться до системи і його можна буде вибрати зі списку на головному вікні додатку. Вікно додання нового вітряка закриється, а користувач повернеться до головного вікна програми.

Для того, щоб розрахувати потужність, що виробляється вітрогенератором, користувачу потрібно вибрати місце на мапі, вітряк, що буде під моніторингом та дати початку та кінця моніторингу. На рисунку 5.5 користувач вибрав місце розташування Ботієвської вітроелектростанції, вітряк Vestas V-112 потужністю 3.4 МВт, що використовується на обраній ВЕС та обрав дати початку та кінця аналізу.

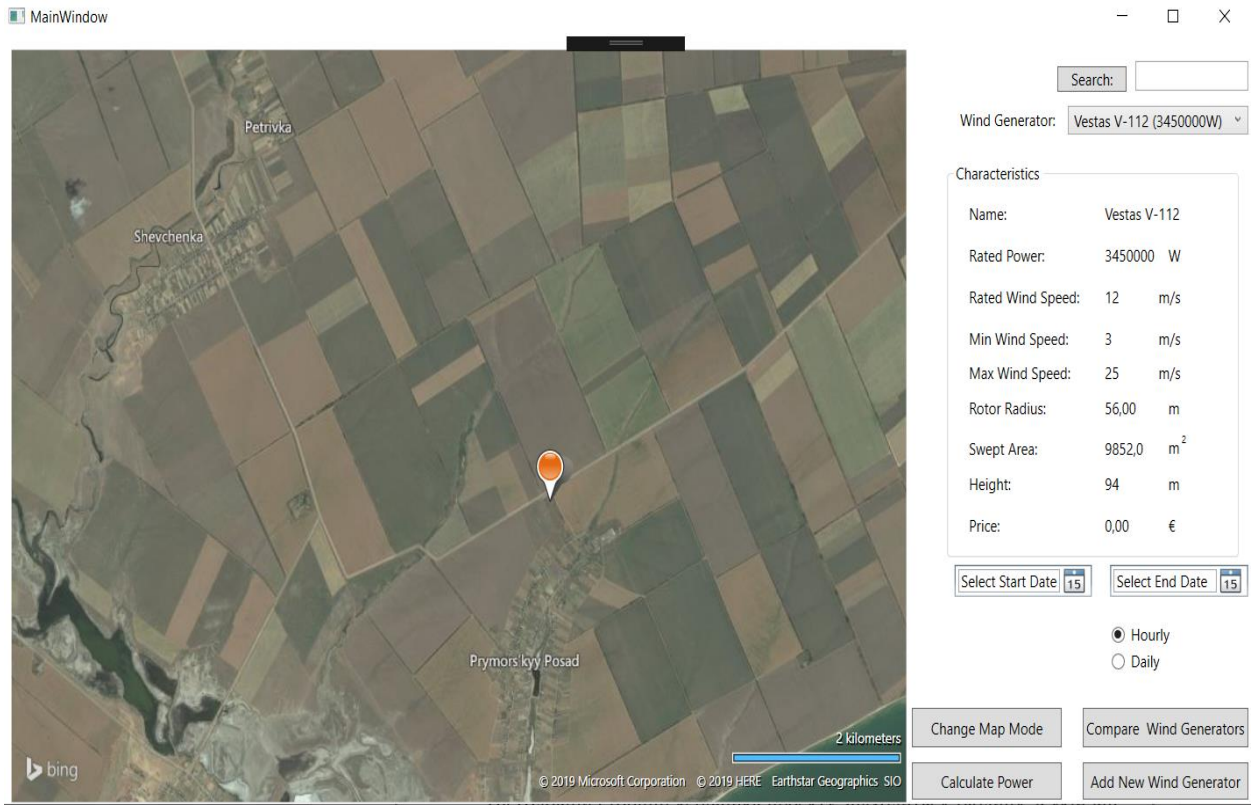


Рисунок 5.5 — Приклад використання програми для моніторингу

Після проведення моніторингу агент згенерував графік генерування енергії обраним вітряком як показано на рисунку 5.6. Ліворуч від графіка знаходиться легенда з датами за якими проводився моніторинг. Вони помічені різними кодами для того, щоб можна було легко побачити якій даті відповідає кожна колонка графіку. По осі Y (вертикальна) показано енергію, що виробив вітряк. По осі X (горизонтальна) години, за які вітряк виробляв енергію.

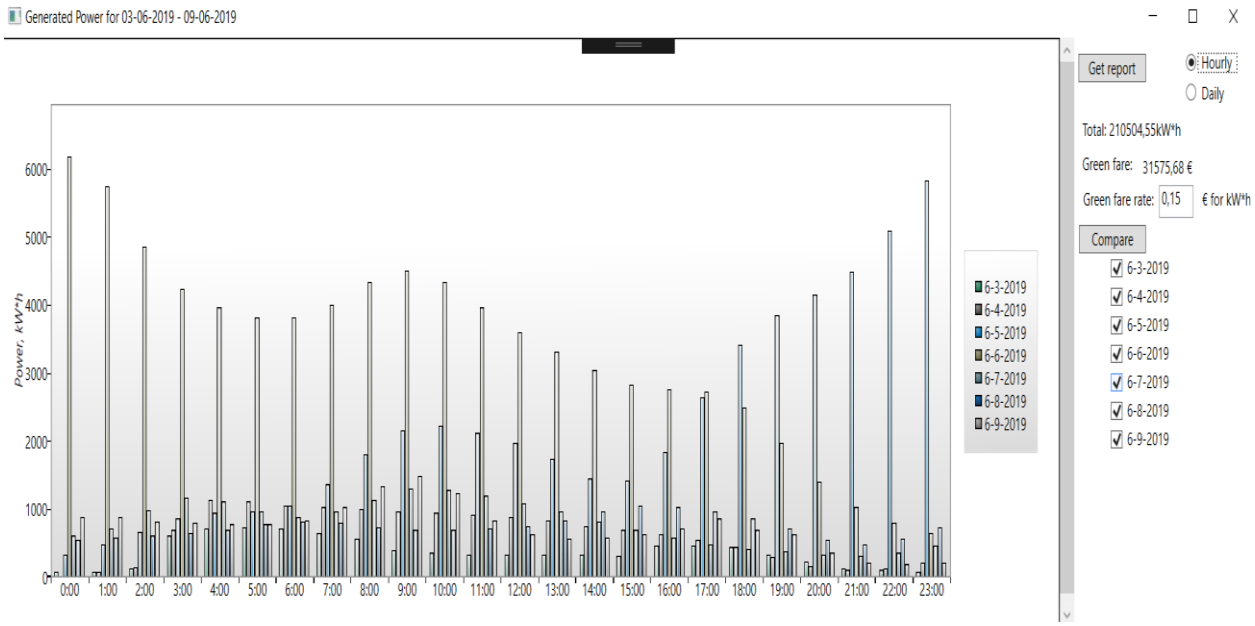


Рисунок 5.6 — Вікно моніторингу вітряка

Користувач може змінити моніторинг з погодинного на добову для моніторингу усереднених результатів роботи вітряка у верхньому правому куту, як показано на рисунку 5.7. У цьому режимі графік показує сумарну електроенергію, що виробив вітряк за добу. Однак ці дані є менш точними, так як швидкість вітру не рахується кожну годину, а береться усереднена. Тому цей режим є сенс використовувати тільки для моніторингу за великий проміжок часу для візуалізації динаміки зміни середньої швидкості вітру за місяць, сезон і т.д.

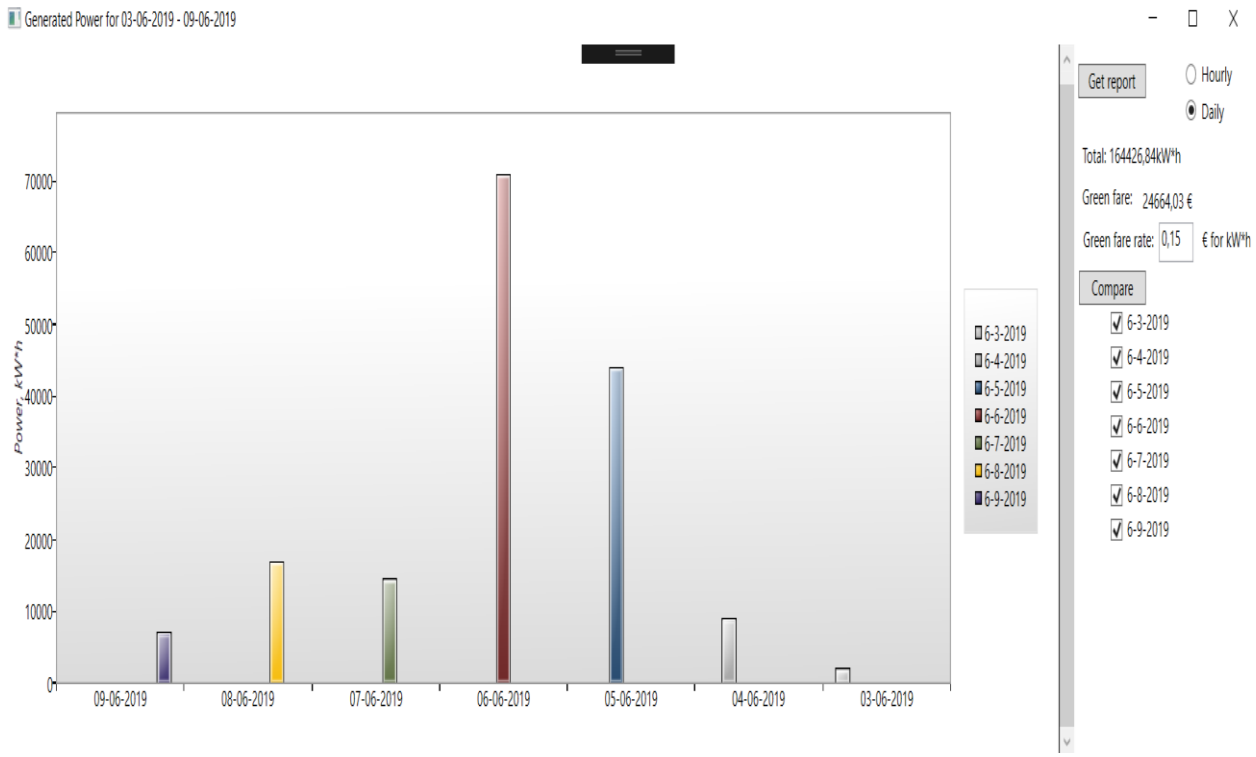


Рисунок 5.7 — Вікно моніторингу вітряка у режимі моніторингу 1 раз на добу

Ліворуч від графіка знаходяться кнопки для отримання звіту, інформація про сумарну вироблену енергію та зелений тариф, кнопка для порівняння роботи вітряків та список дат, за які проводився моніторинг. При деактивації однієї з дат графік виключає її зі списку та перераховує розрахунки. Так користувач може подивитись результати роботи не тільки за весь тиждень, а й за кожний окремий день, або тільки за перший та останній день моніторингу. Такий функціонал надає користувачу додатку можливість проаналізувати інформацію так, як саме йому (користувачу) потрібно. З рисунку 5.8 видно, що за обраний день (06.06.2019) найбільш продуктивно вітряк працював протягом першої години дня, згенерувавши 6190 КВт*год енергії, що можна побачити навівши мишку на колонку графіка, сумарно за цей день вітряком було вироблено 80262 КВт*год енергії, що еквівалентно 12 тисячам євро при тарифі 0.15 € за КВт*год.

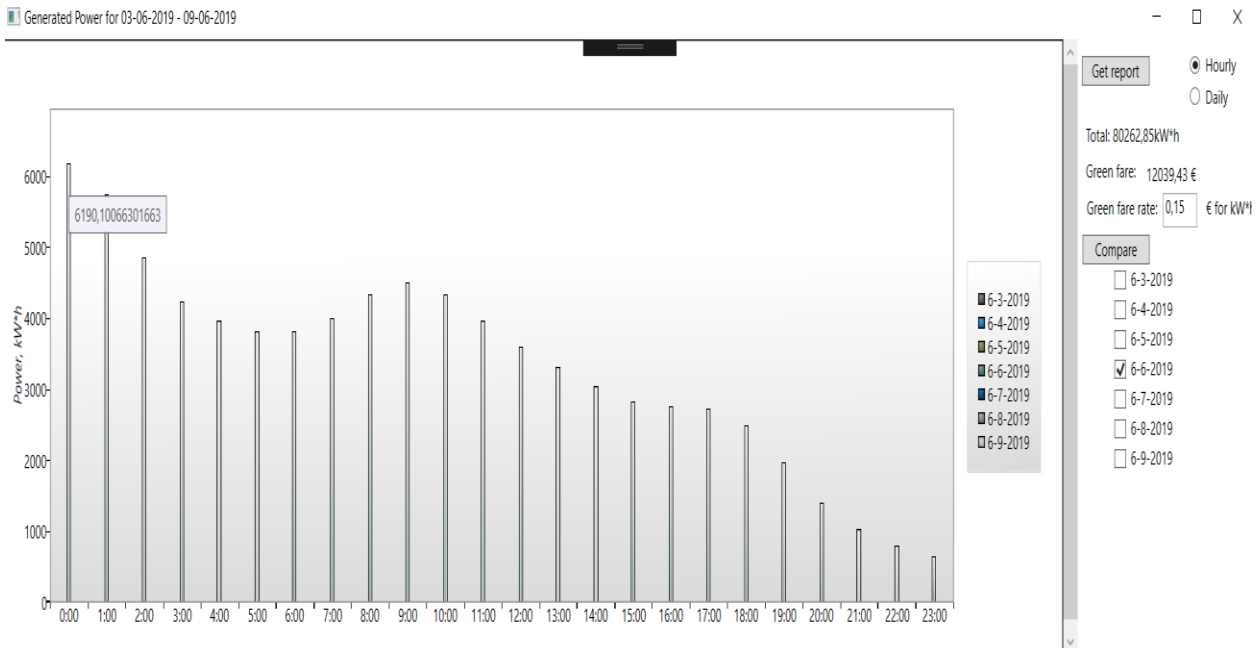


Рисунок 5.8 — Графік моніторингу вітроелектроенергії за обраний день

При натисканні на клавішу “Get report” (отримати звіт) користувач може вибрати спосіб отримання звіту (рисунок 5.9):

— Роздрукувати, обравши принтер;

— Зберегти як PDF файл;

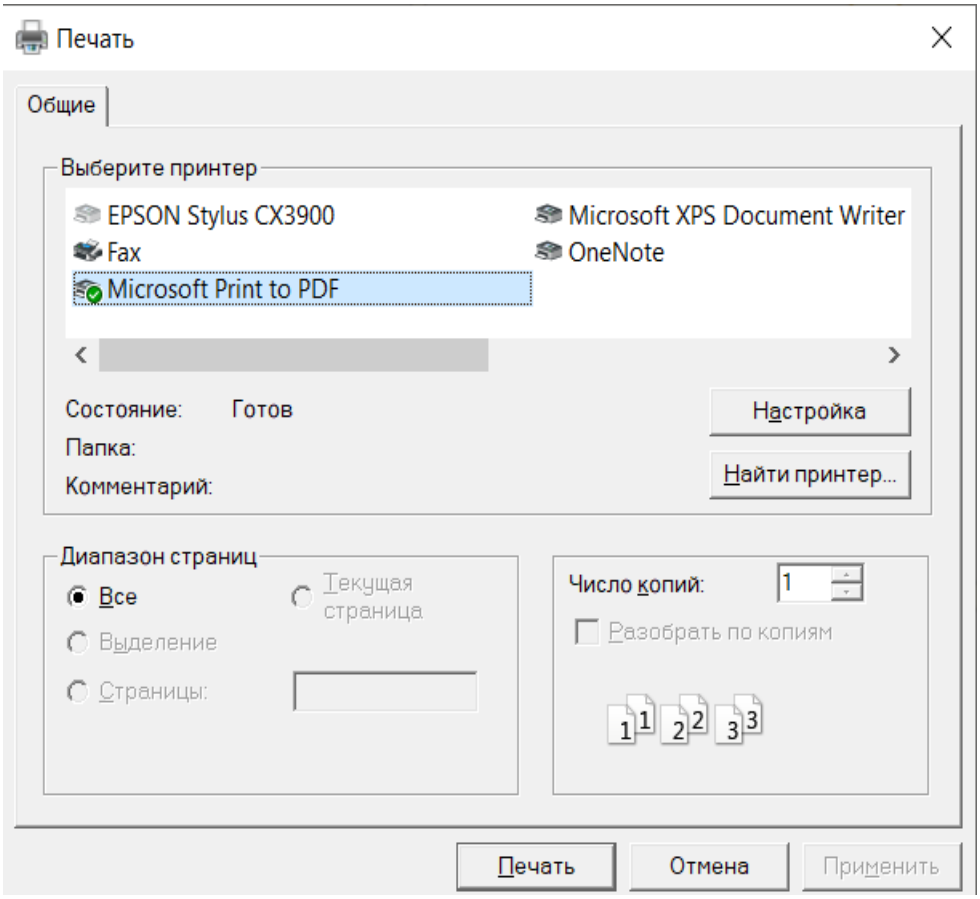


Рисунок 5.9 — Вікно для отримання звіту з роботи вітряка

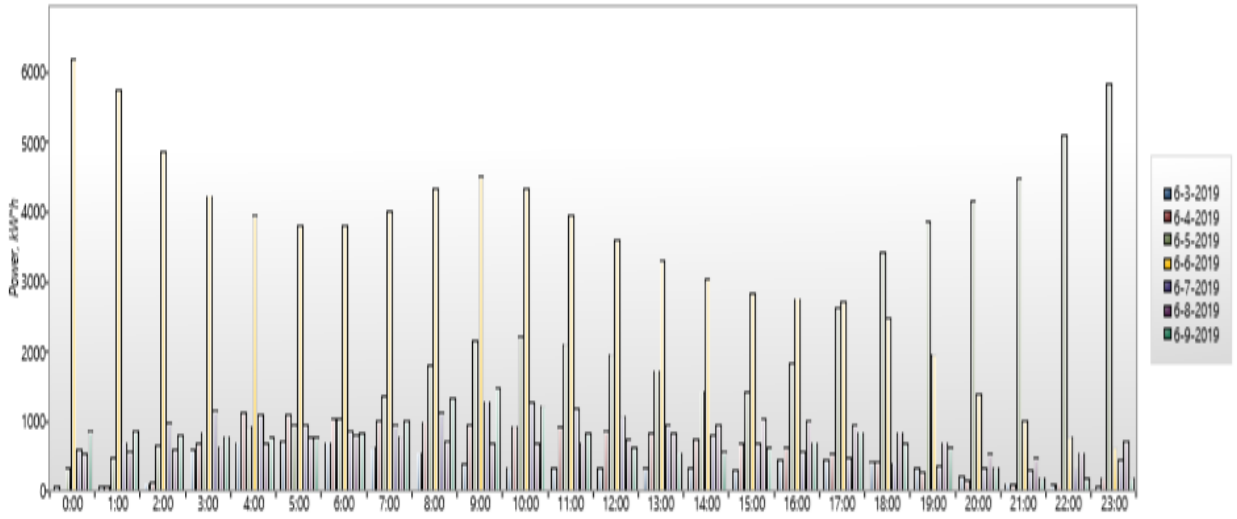
При збереженні звіту як PDF файл користувач може дати назву документу та зберегти його у обраному місці на локальному комп’ютері. Так документ можна відправити електронною поштою або роздрукувати.

У звіті зберігається графік моніторингу виробленої електроенергії вітряком та повна інформація за кожну годину або день коли вітряк працював. У таблиці під графіком відображена така інформація:

— Назва вітряка;
— Дата моніторингу;
— Година дня моніторингу (якщо обрано погодинний режим);
— Швидкість вітру за годину (добу);
— Вироблена електроенергія за годину (добу);
— Інформація про помилки у роботі вітряка;
— Номінальна потужність обраного вітряка;

— Номінальна швидкість вітру обраного вітряка;
— Місце розташування ВЕС;
— Сумарна вироблена електроенергія кожні за добу;
— Зелений тариф;
— Ціна електроенергії за зеленим тарифом;

Рядок з годиною (днем) протягом якої була помилка у роботі вітряка підсвічується червоним кольором, щоб користувачу було легко побачити коли сталася несправність та її причину. На рисунку 5.10 червоним кольором виділено 12 годину ночі 04.06.2019 та вказано, що причиною помилки є замала швидкість вітру, через що вітряк не міг функціонувати. Також при досягненні номінальної швидкості вітру рядок підсвічується зеленим кольором для того, щоб користувач додатку міг порівняти реальне вироблення електроенергії з номінальним та переконатися, що все працює згідно зі стандартами виробника вітряка.



Name	Date	Hour	Wind Speed (m/s)	Power (kW*h)	Error	Rated Power (W)	Rated Wind Speed (m/s)	Location	Total (kW*h)	Green Rate (€/kW*h)	Green Fare (€)
estas V-112	3 июня 2019 г.	0:00:00	2,09	66,33		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	9,95
estas V-112	3 июня 2019 г.	1:00:00	2,13	70,21		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	10,53
estas V-112	3 июня 2019 г.	2:00:00	2,55	120,47		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	18,07
estas V-112	3 июня 2019 г.	3:00:00	4,36	602,19		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	90,33
estas V-112	3 июня 2019 г.	4:00:00	4,61	711,83		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	106,77
estas V-112	3 июня 2019 г.	5:00:00	4,64	725,82		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	108,87
estas V-112	3 июня 2019 г.	6:00:00	4,58	698,02		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	104,70
estas V-112	3 июня 2019 г.	7:00:00	4,46	644,58		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	96,69
estas V-112	3 июня 2019 г.	8:00:00	4,26	561,70		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	84,25
estas V-112	3 июня 2019 г.	9:00:00	3,76	386,22		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	57,93
estas V-112	3 июня 2019 г.	10:00:00	3,63	347,53		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	52,13
estas V-112	3 июня 2019 г.	11:00:00	3,58	333,37		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	50,00
estas V-112	3 июня 2019 г.	12:00:00	3,53	319,59		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	47,94
estas V-112	3 июня 2019 г.	13:00:00	3,53	319,59		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	47,94
estas V-112	3 июня 2019 г.	14:00:00	3,54	322,32		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	48,35
estas V-112	3 июня 2019 г.	15:00:00	3,44	295,77		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	44,36
estas V-112	3 июня 2019 г.	16:00:00	3,94	444,39		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	66,66
estas V-112	3 июня 2019 г.	17:00:00	3,98	458,06		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	68,71
estas V-112	3 июня 2019 г.	18:00:00	3,87	421,12		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	63,17
estas V-112	3 июня 2019 г.	19:00:00	3,55	325,06		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	48,76
estas V-112	3 июня 2019 г.	20:00:00	3,10	216,45		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	32,47
estas V-112	3 июня 2019 г.	21:00:00	2,58	124,78		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	18,72
estas V-112	3 июня 2019 г.	22:00:00	2,30	88,40		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	13,26
estas V-112	3 июня 2019 г.	23:00:00	2,11	68,25		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	10,24
estas V-112	4 июня 2019 г.	0:00:00	2,05	0,00	Wind Speed is below the permissable rate	3450000	12	Pryazovskyi raion, 72401, Ukraine	8672,04	0,15	0,00
estas V-112	4 июня 2019 г.	1:00:00	2,26	83,87		3450000	12	Pryazovskyi raion, 72401, Ukraine		0,15	12,58

Рисунок 5.10 — Приклад звіту з роботи агента

Для порівняння роботи двох вітряків у вікні моніторингу вітряка користувачу потрібно натиснути кнопку “Compare” (порівняти). Таким чином користувач опиниться на головному вікні додатку, де зможе обрати другий вітряк для порівняння. Порівнювати можна вітряки різних типів, на різних місцевостях за різні проміжки часу, але найбільш точним порівнянням буде збереження дат та місця проведення моніторингу якщо ціллю порівняння є знаходження найбільш ефективного вітряка у заданій локації. Однак, якщо користувач хоче порівняти ефективність роботи одного й того ж вітряка у різних локаціях, то він може обрати той самий вітряк, але інше місце для проведення моніторингу. Після вибору другого вітряка для порівняння, дат та місця проведення моніторингу користувачу потрібно натиснути кнопку “Calculate Power” для того, щоб здійснити моніторинг з новими параметрами. Якщо змінено лише тип вітряка, а місце та час моніторингу залишились без змін з попереднього моніторингу, то агент бере кешовані дані швидкості вітру та аналізує результати з новим вітряком. Таким чином підвищується швидкодія додатку та зменшується кількість звертань до прикладного програмного інтерфейсу DarkSky.

Після того як агент проаналізує дані для нового вітряка у вікні моніторингу користувач може натиснути кнопку “Compare to” <назва першого вітряка> (рисунок 5.11).

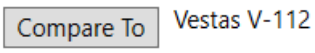


Рисунок 5.11 — Кнопка для порівняння роботи двох вітряків

Після кліку по кнопці відкриється вікно порівняння роботи двох вітряків. У вікні знаходиться інформація про такі показники:

— Назва вітряка;
— Місце (місця) проведення моніторингу;
— Номінальна потужність кожного вітряка та їх різниця;
— Номінальна швидкість вітру та їх різниця;
— Сумарна кількість електроенергії, що виробили вітряки та їх різниця;
— Усереднена кількість виробленої електроенергії (за годину або добу) та їх

різниця;

— Середня швидкість вітру за час моніторингу двох вітряків та їх різниця;
— Висота вітряків та їх різниця;
— Ціна вітряків та їх різниця;

На рисунку 5.12 порівнюються вітряки Vestas V-112 та Enercon E40. Як видно з рисунку номінальна потужність першого вітряка набагато більше, що сприяє більшому середньому виробленню електроенергії. Також висота вежі першого вітряка більше, тому й середня швидкість вітру більше. Це зумовлено тим, що швидкість вітру збільшується з висотою, тому програмний агент розраховує швидкість вітру беручи до уваги висоту вежі вітряка. Різниця між показниками роботи вітряків знаходиться у середній колонці, та підсвічується зеленим кольором якщо показник першого вітряка кращий та червоним, якщо показник нижче.

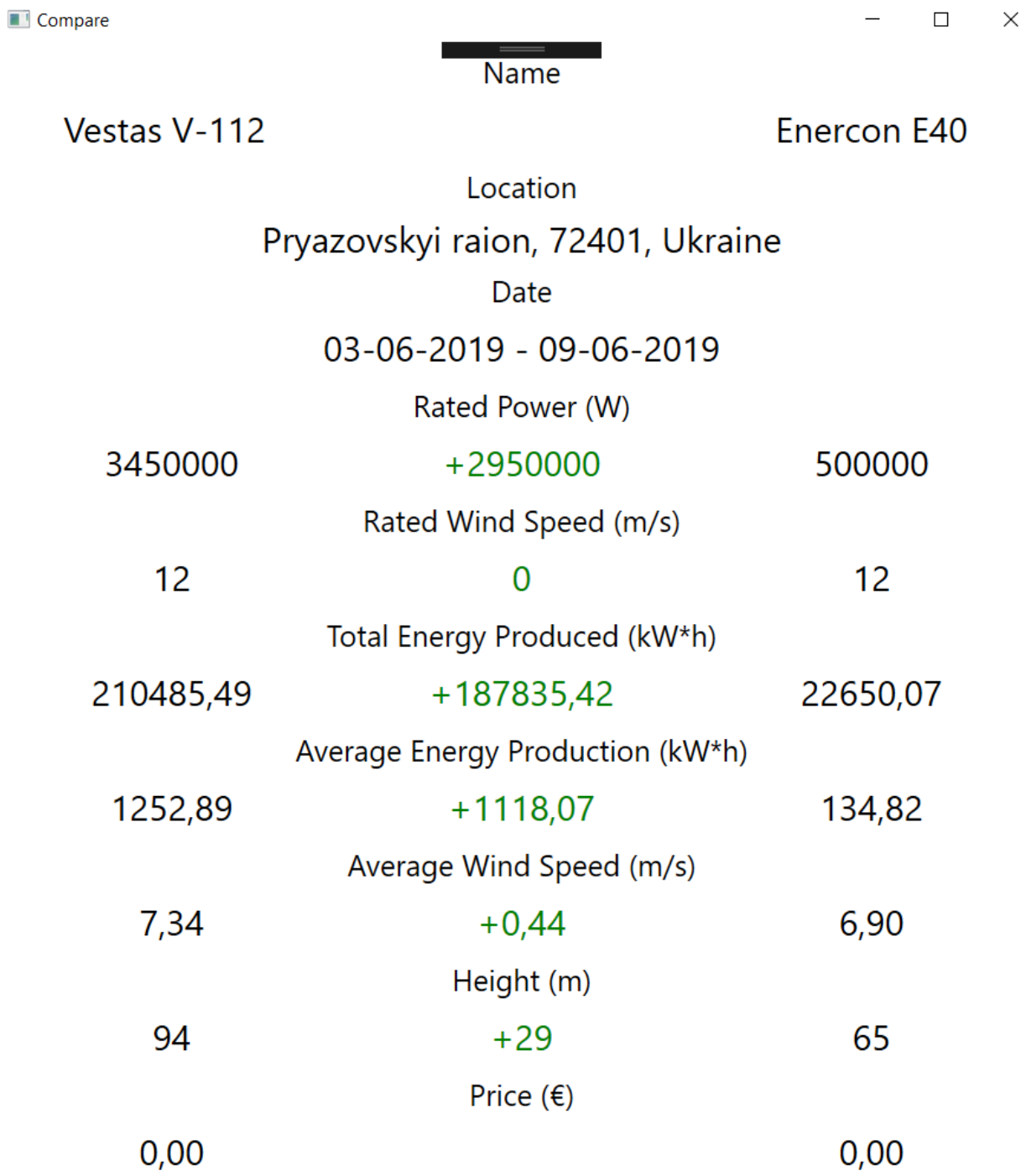


Рисунок 5.12 — Вікно порівняння роботи вітряків

Окрім порівняння роботи різних вітряків на одній ВЕС, агент надає змогу порівнювати вітряки одного типу на різних місцевостях для можливості обрання найбільш оптимального місця розташування ВЕС або порівняння функціонування різних ВЕС.

Для прикладу на рисунку 5.13 було обрано 2 працюючі вітроелектростанції:

Ботієвська ВЕС та Очаківська ВЕС за період з 01.06.2019 по 30.06.2019.

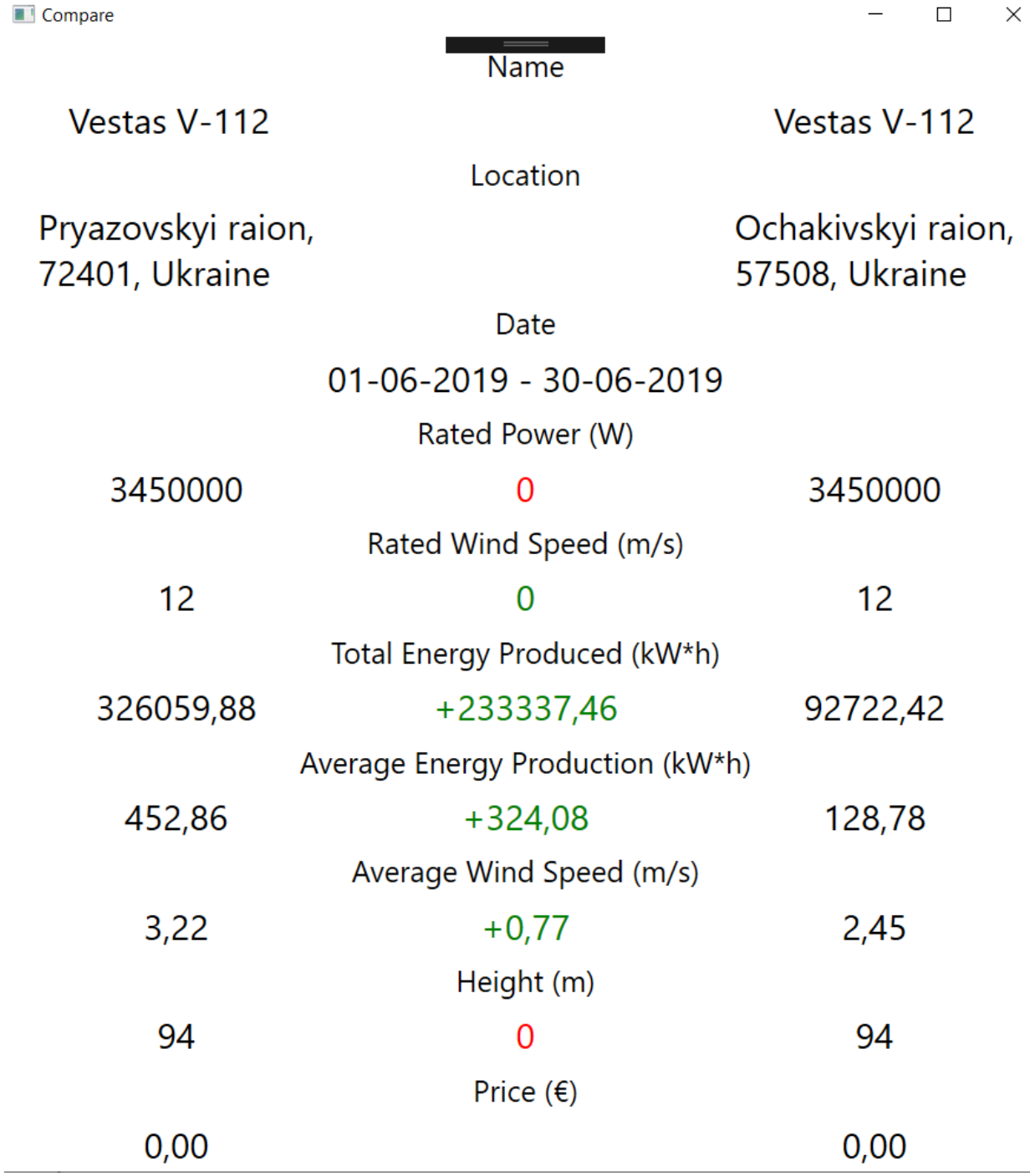


Рисунок 5.13 — Вікно порівняння вітряків на різних місцях

Таким чином програмний агент не тільки аналізує та розраховує вироблену електроенергію, а й допомагає у розробці ВЕС, надаючи змодельовану інформацію про вітропотенціал обраної місцевості.

Якщо користувач не знає який вітряк обрати або хоче порівняти характеристики

двох вітряків для впровадження можна скористатись функцією порівняння технічних характеристик вітряків.

Для цього на головному вікні додатку користувачу треба натиснути кнопку “Compare Wind Generators” (порівняти вітряні генератори). Після кліку опиниться вікно порівняння вітряків. Характеристики за якими порівнюються вітряки:

— Номінальна потужність;

— Номінальна швидкість вітру;

— Мінімальна та максимальна швидкості вітру при яких вітряк може функціонувати;

— Радіус ротора;

— Ометаєма площа;

— Висота вежі;

— Ціна;

Як і у вікні порівняння роботи вітряків у цьому вікні відображено таблицю з двома вітряками. Згори над кожним вітряком розташовано списки всіх вітряків, що є у базі даних. За допомогою вибору вітряків зі списків користувач може порівняти декілька вітряків обираючи їх по черзі.

На рисунку 5.14 наведено приклад порівняння двох вітряків. Як видно з рисунку кращі характеристики підсвічуються зеленим кольором для того, щоб користувачу було легше виявити кращий вітряк. Так, наприклад, вітряк Enercon E30 має висоту вежі вищу за Enercon E40, що може суттєво впливати на вироблювану електроенергію, але коштує більше другого.

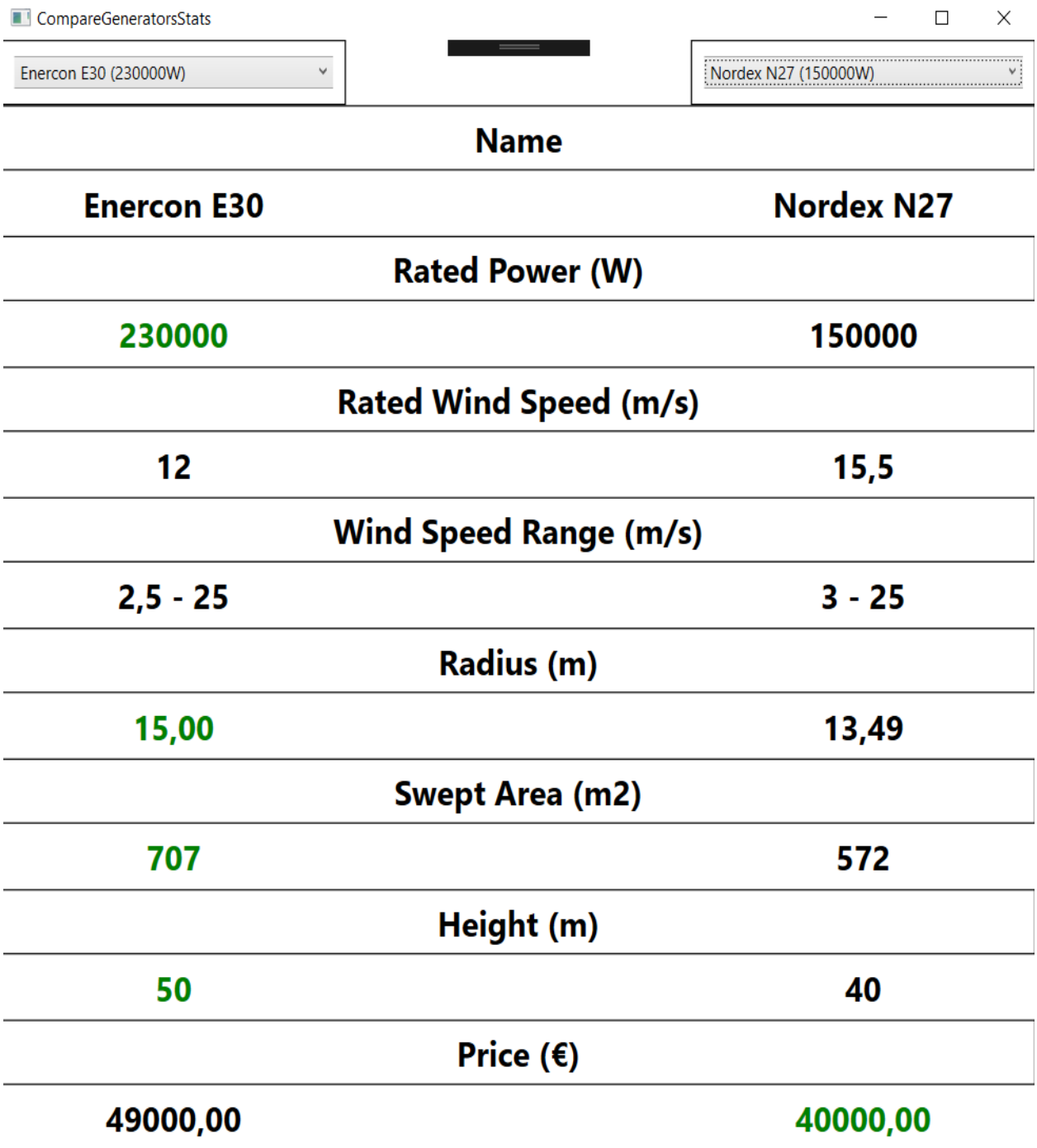


Рисунок 5.14 — Вікно порівняння технічних характеристик двох вітряків

ВИСНОВКИ

В процесі виконання роботи було виконано наступне:

1. Отримано необхідні знання для створення програмного агента моніторингу та управління вітроенергетичною станцією.

2. Сформульована проблематика розробки програмного агента автоматизації моніторингу та управління вітроенергетичної станції. Для реалізації поставленої задачі було вирішено створити клієнтський додаток з локальною базою даних, адже завдяки такому підходу користувачі отримають гнучку систему, яка дозволяє самостійно проводити аналіз вітропотенціалу для будь-якої точки планети та проводити моніторинг з точними вхідними даними.

3. Розроблено програмний агент, який дозволяє проводити моніторинг та управління вітряною електростанцією, оптимізацію роботи вітряків, порівняння та знаходження оптимальної місцевості для будування вітряної електростанції, робити порівняння вітряків різної потужності для роботи на заданій місцевості або порівняння вітропотенціалу для різних місць використавши сучасні програмно-технічні рішення для створення десктопного додатку розробленого на мові програмування високого рівня C# за допомогою інтегрованої середи розробки Visual Studio.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Кремерс Енріке Моделювання та симуляція систем електричної енергії через комплексний системний підхід з використанням агентних моделей. 2013. Видавництво KIT Scientific.

2. Вітроенергетика / за ред. Д. де Рензо / пер. з англ. Зубарева В.В. [за ред. Шефтер Я.І.]. Москва, 1982.

3. Сайт GE Renewable Energy // Wind Plant Wake Managament [Електронний ресурс] – Режим доступу: https://www.ge.com/renewableenergy/wind-energy/technology/wind-plant-wake-management/Дата доступу: 03.06.2019

4. Сайт Power Electronics // Power Managament Chapter 16: Wind Power. [Електронний ресурс] - Режим доступу: https://www.powerelectronics.com/power-management/power-management-chapter-16-wind-power/Дата доступу: 03.06.2019

5. Документація MSSql [Електронний ресурс] — https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation/ Дата доступу: 02.06.2019

6. Мангіна Е.Е Огляд програмних продуктів для мультиагентних систем/Review of software products for multi-agent systems — Видавництво IEEE, 2001

7. Аксой Х., Фуат Топрак З., Айтек А., Ердем Нал Н. Стохастичне генерування даних середньої швидкості вітру/ Stochastic generation of hourly mean wind speed data — Відновлювана Енергія, випуск 29(14), 2004

8. Уілліс Х.Л. Розподілене виробництво електроенергії: планування та оцінка/Distributed power generation: planning and evaluation — Marcel Dekker, New York, 2000

9. Тімбус А., Ларссон М., Юен С. Активне управління розподіленими енергетичними ресурсами з використанням стандартизованих комунікацій та сучасних інформаційних технологій/Active Management of Distributed Energy Resources Using Standardized Communications and Modern Information Technologies — IEEE, 2009

10. Пуджіанто Д., Рамзай С. Віртуальна електростанція та системна інтеграція

розподілених енергоресурсів/Virtual power plant and system integration of distributed energy resources — IET Renewable Power Generation, vol 1,2007

11. Фаулер М. Рефакторинг. Улучшение существующего кода [Електронний ресурс] / М. Фаулер. — Пер. с англ. — СПб: Символ-Плюс, 2003. — 432 с. — Режим доступу: dolina-sun.ru/filemanager/download/77

12. К. Дж. Дейт. Введение в системы баз данных / An Introduction to Database Systems. — 7-е изд. — «Вильямс», 2001. — 1092 с.

13. Офіційний сайт Visual Studio [Електронний ресурс] — Режим доступу: https://visualstudio.microsoft.com/Дата доступу: 03.06.2019

Додаток 1

Програмний агент управління та моніторингу вітроенергетичної установки

Специфікація

УКР.НТУУ“КПІ”.ТМ51106_19Б

Аркушів 2

2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ51106_19Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ». ТМ51106_19Б 12-1	Текст програмного модулю	
УКР.НТУУ«КПІ». ТМ51106_19Б 13-1	Опис програми	

Додаток 2

Програмний агент управління та моніторингу вітроенергетичної установки

Текст програмного модулю

УКР.НТУУ“КПІ”.ТМ51106_19Б 12-1

Аркушів 10

2019

```
namespace WindEnergy
{
    /// <summary>
    /// Interaction logic for ChartWindow.xaml
    /// </summary>
    public partial class ChartWindow : Window
    {
        MainWindow main;
        double totalPower;
        public List<Weather> weatherList;
        List<CheckBox> checkBoxes;
        List<DailyTotalPower> dailyTotalPowers;
        double modifiableTotalPower;
        double greenFareRate = 0;
        public ChartWindow()
        {
            InitializeComponent();
            main = (MainWindow)Application.Current.MainWindow;
            checkBoxes = new List<CheckBox>();
            weatherList = new List<Weather>();
            dailyTotalPowers = new List<DailyTotalPower>();
            modifiableTotalPower = 0;
            chart.Loaded += Chart_Loaded;
            totalPower = 0;
            GetGreenFareRate();
            if (main.startDate != main.endDate)
                Title = "Generated Power for " + main.startDate.ToShortDateString() +
" - " + main.endDate.ToShortDateString();
            else
                Title = "Generated Power for " + main.startDate.ToShortDateString();

            if (main.comparisonOn)
            {
                compareBtn.Content = "Compare To";
                comparerLbl.Content =
main.firstWeatherToCompare[0].WindGenerator.Name;
            }
            else
            {
                compareBtn.Content = "Compare";
                comparerLbl.Content = string.Empty;
            }
        }

        private void GetGreenFareRate()
        {
            //greenRateTB.Text = greenRateTB.Text.Replace(',', '.');
            bool success = double.TryParse(greenRateTB.Text, out greenFareRate);
            if (!success)
            {
                MessageBox.Show("Enter valid Green Fare Rate");
            }
        }

        private void Item_Checked(object sender, RoutedEventArgs e)
        {
            var item = chart.Series.Where(x => ((ColumnSeries)x).Title ==
((CheckBox)sender).Content).FirstOrDefault();
            ((ColumnSeries)item).Visibility = Visibility.Visible;
            double dayPower = dailyTotalPowers.Where(x => x.Date.Date ==
(((CheckBox)sender).Content as DateTime?))
```

```
                                                .Select(p =>
p.TotalPower).FirstOrDefault();
            modifiableTotalPower += dayPower;
            totalLbl.Content = "Total: " + modifiableTotalPower.ToString("0.00") +
"kW*h";
            greenFareLbl.Content = (modifiableTotalPower *
greenFareRate).ToString("0.00") + " €";
        }

        private void CheckBox_Unchecked(object sender, RoutedEventArgs e)
        {
            var item = chart.Series.Where(x => ((ColumnSeries)x).Title ==
((CheckBox)sender).Content).FirstOrDefault();
            ((ColumnSeries)item).Visibility = Visibility.Hidden;
            double dayPower = dailyTotalPowers.Where(x => x.Date.Date ==
(((CheckBox)sender).Content as DateTime?))
                                                .Select(p =>
p.TotalPower).FirstOrDefault();
            modifiableTotalPower -= dayPower;
            totalLbl.Content = "Total: " + modifiableTotalPower.ToString("0.00") +
"kW*h";
            greenFareLbl.Content = (modifiableTotalPower *
greenFareRate).ToString("0.00") + " €";
        }

        private void Chart_Loaded(object sender, RoutedEventArgs e)
        {
            GetPowerValues();
        }

        public void GetPowerValues()
        {
            dailyTotalPowers.Clear();
            legend.Children.Clear();
            checkBoxes.Clear();
            if ((bool)main.dailyRB.IsChecked)
                weatherList = main.weatherListDaily;
            if ((bool)main.hourlyRB.IsChecked)
                weatherList = main.weatherListHourly;
            chart.Series.Clear();
            List<int> dates = main.weatherListHourly.Select(x =>
x.date.DayOfYear).Distinct().ToList();
            totalPower = 0;
            foreach (var day in dates)
            {
                List<KeyValuePair<string, double>> dataList = new
List<KeyValuePair<string, double>>();
                foreach (var weather in weatherList.Where(x => x.date.DayOfYear ==
day))
                {
                    if (weather.date.DayOfYear == day)
                    {
                        double power = 0;
                        if ((bool)main.hourlyRB.IsChecked)
                        {
                            power =
main.selectedWindGen.CalculatePowerForHour(weather.windSpeed);
                            weather.WindGenerator = InputGeneratorInfo(power);
                            dataList.Add(new KeyValuePair<string,
double>(weather.date.ToShortTimeString(), power));
                        }
                        else if ((bool)main.dailyRB.IsChecked)
```

```
                        {
                            power =
main.selectedWindGen.CalculatePowerForDay(weather.windSpeed);
                            weather.WindGenerator = InputGeneratorInfo(power);
                            dataList.Add(new KeyValuePair<string,
double>(weather.date.ToShortDateString(), power));
                        }

                        totalPower += power;
                    }
                }
                ColumnSeries series = new ColumnSeries()
                {
                    Title = weatherList.Where(x => x.date.DayOfYear == day).Select(d
=> d.date).FirstOrDefault(),
                    IndependentValuePath = "Key",
                    DependentValuePath = "Value",
                    ItemsSource = dataList,

                };
                dailyTotalPowers.Add(new DailyTotalPower()
                {
                    Date = weatherList.Where(x => x.date.DayOfYear == day).Select(d
=> d.date).FirstOrDefault(),
                    TotalPower = totalPower
                });
                totalPower = 0;
                CheckBox checkBox = new CheckBox();
                checkBox.IsChecked = true;
                checkBox.Content = series.Title;
                checkBox.Checked += Item_Checked;
                checkBox.Unchecked += CheckBox_Unchecked;
                checkBoxes.Add(checkBox);
                ColumnDefinition column = new ColumnDefinition();
                RowDefinition row = new RowDefinition();
                column.Width = new GridLength(200);
                row.Height = new GridLength(20);
                legend.ColumnDefinitions.Add(column);
                legend.RowDefinitions.Add(row);
                Grid.SetRow(checkBox, checkBoxes.Count - 1);
                Grid.SetColumn(checkBox, 0);

                legend.Children.Add(checkBox);
                chart.Series.Add(series);
            }
            legend.Height = checkBoxes.Count * 20;
            totalPower = dailyTotalPowers.Select(x => x.TotalPower).Sum();
            modifiableTotalPower = totalPower;
            totalLbl.Content = "Total: " + totalPower.ToString("0.00") + "kW*h";
            greenFareLbl.Content = (totalPower * greenFareRate).ToString("0.00") + "
€";
        }

        WindGenerator InputGeneratorInfo(double power)
        {
            return new WindGenerator()
            {
                Power = power,
                Name = main.selectedWindGen.Name,
                ErrorMessage = main.selectedWindGen.ErrorMessage,
                RatedPower = main.selectedWindGen.RatedPower,
                RatedWindSpeed = main.selectedWindGen.RatedWindSpeed,
```

```
            WindSpeed = main.selectedWindGen.WindSpeed,
            Height = main.selectedWindGen.Height,
            Price = main.selectedWindGen.Price
        };
    }

    public void CheckAllDates()
    {
        foreach (var item in legend.Children)
        {
            if (item is CheckBox)
            {
                ((CheckBox)item).IsChecked = true;
            }
        }
    }

    private void DailyRB_Checked(object sender, RoutedEventArgs e)
    {
        main.dailyRB.IsChecked = dailyRB.IsChecked;
        CheckAllDates();
        GetPowerValues();
    }

    private void HourlyRB_Checked(object sender, RoutedEventArgs e)
    {
        main.hourlyRB.IsChecked = hourlyRB.IsChecked;
        CheckAllDates();
        GetPowerValues();
    }

    private void ReportBtn_Click(object sender, RoutedEventArgs e)
    {
        UIElement toDelete = new UIElement();
        if (printCanvas.RowDefinitions.Count > 1)
        {
            printCanvas.RowDefinitions.RemoveAt(1);

            foreach (UIElement item in printCanvas.Children)
            {
                if (Grid.GetRow(item) == 1)
                    toDelete = item;
            }
            printCanvas.Children.Remove(toDelete);
        }
        Grid grid = new Grid();
        RowDefinition row = new RowDefinition()
        {
            Height = new GridLength(0, GridUnitType.Star)
        };
        printCanvas.RowDefinitions.Add(row);
        grid.Children.Add(GenerateGrid());
        Grid.SetRow(grid, 1);
        Grid.SetColumn(grid, 0);
        printCanvas.Children.Add(grid);
        scroller.VerticalScrollBarVisibility = ScrollBarVisibility.Visible;
        PrintDialog printDialog = new PrintDialog();
        scroller.Content = printCanvas;
        scroller.ScrollToTop();
        printDialog.PageRangeSelection = PageRangeSelection.AllPages;
        if (printDialog.ShowDialog() == true)
        {
```

```
            printDialog.UserPageRangeEnabled = true;
            printDialog.PrintVisual(scroller.Content as Visual, "Printing");
        }
    }

    private Grid GenerateGrid()
    {
        DateTime? currentDate = null;
        double dailyTotal = 0;
        //Margin
        Thickness thickness = new Thickness(5, 5, 5, 5);

        Grid docGrid = new Grid();
        RowDefinition row1 = new RowDefinition();
        row1.Height = new GridLength(40);
        ColumnDefinition column = new ColumnDefinition();
        column.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column1 = new ColumnDefinition();
        column1.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column2 = new ColumnDefinition();
        column2.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column3 = new ColumnDefinition();
        column3.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column4 = new ColumnDefinition();
        column4.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column5 = new ColumnDefinition();
        column5.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column6 = new ColumnDefinition();
        column6.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column7 = new ColumnDefinition();
        column7.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column8 = new ColumnDefinition();
        column8.Width = new GridLength(350, GridUnitType.Auto);
        ColumnDefinition column9 = new ColumnDefinition();
        column9.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column10 = new ColumnDefinition();
        column10.Width = new GridLength(250, GridUnitType.Auto);
        ColumnDefinition column11 = new ColumnDefinition();
        column11.Width = new GridLength(250, GridUnitType.Auto);

        docGrid.RowDefinitions.Add(row1);
        docGrid.ColumnDefinitions.Add(column);
        docGrid.ColumnDefinitions.Add(column1);
        docGrid.ColumnDefinitions.Add(column2);
        docGrid.ColumnDefinitions.Add(column3);
        docGrid.ColumnDefinitions.Add(column4);
        docGrid.ColumnDefinitions.Add(column5);
        docGrid.ColumnDefinitions.Add(column6);
        docGrid.ColumnDefinitions.Add(column7);
        docGrid.ColumnDefinitions.Add(column8);
        docGrid.ColumnDefinitions.Add(column9);
        docGrid.ColumnDefinitions.Add(column10);
        docGrid.ColumnDefinitions.Add(column11);

        TextBlock name = new TextBlock();
        name.Text = "Name";
        TextBlock date = new TextBlock();
        date.Text = "Date";
        TextBlock power = new TextBlock();
        power.Text = "Power (kW*h)";
        TextBlock wind = new TextBlock();
        wind.Text = "Wind Speed (m/s)";
```

```
            TextBlock error = new TextBlock();
            error.Text = "Error";
            TextBlock hours = new TextBlock();
            hours.Text = "Hour";
            TextBlock rPower = new TextBlock();
            rPower.Text = "Rated Power (W)";
            TextBlock rWind = new TextBlock();
            rWind.Text = "Rated Wind Speed (m/s)";
            TextBlock location = new TextBlock();
            location.Text = "Location";
            TextBlock total = new TextBlock();
            total.Text = "Total (kW*h)";
            TextBlock rate = new TextBlock();
            rate.Text = "Green Rate (€/kW*h)";
            TextBlock totalGreen = new TextBlock();
            totalGreen.Text = "Green Fare (€)";

            List<TextBlock> headers = new List<TextBlock>();
            headers.Add(name);
            headers.Add(date);
            headers.Add(hours);
            headers.Add(wind);
            headers.Add(power);
            headers.Add(error);
            headers.Add(rPower);
            headers.Add(rWind);
            headers.Add(location);
            headers.Add(total);
            headers.Add(rate);
            headers.Add(totalGreen);

            for (int i = 0; i < headers.Count; i++)
            {
                headers[i].HorizontalAlignment = HorizontalAlignment.Center;
                headers[i].Margin = new Thickness(5, 5, 5, 5);
                Grid.SetRow(headers[i], 0);
                Grid.SetColumn(headers[i], i);
                docGrid.Children.Add(headers[i]);
            }

            for (int i = 0; i <= headers.Count; i++)
            {
                Rectangle rectangle = new Rectangle();
                rectangle.Stroke = Brushes.Black;
                rectangle.Fill = Brushes.Transparent;
                Grid.SetRow(rectangle, 0);
                Grid.SetColumn(rectangle, i);
                docGrid.Children.Add(rectangle);
            }


            for (int i = 0; i < weatherList.Count; i++)
            {
                if (currentDate == null)
                    currentDate = weatherList[i].date;
                if (currentDate != weatherList[i].date.Date || (i + 1) ==
weatherList.Count)
                {
                    RowDefinition dailyTotalRow = new RowDefinition();
                    dailyTotalRow.Height = new GridLength(30, GridUnitType.Auto);
                    docGrid.RowDefinitions.Add(dailyTotalRow);
                    ColumnDefinition columnDailyTotal = new ColumnDefinition();
```

```
    docGrid.ColumnDefinitions.Add(columnDailyTotal);
    TextBlock totalDailyNum = new TextBlock();
    totalDailyNum.FontWeight = FontWeights.Bold;
    totalDailyNum.Text = dailyTotal.ToString("0.00");
    totalDailyNum.HorizontalAlignment = HorizontalAlignment.Center;
    totalDailyNum.Margin = thickness;
    Grid.SetRow(totalDailyNum, i + 1);
    Grid.SetColumn(totalDailyNum, 9);
    docGrid.Children.Add(totalDailyNum);

    dailyTotal = 0;
    currentDate = weatherList[i].date;
}

RowDefinition row = new RowDefinition();
row.Height = new GridLength(30);
docGrid.RowDefinitions.Add(row);
ColumnDefinition columnName = new ColumnDefinition();
columnName.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnDate = new ColumnDefinition();
columnDate.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnHour = new ColumnDefinition();
columnHour.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnWind = new ColumnDefinition();
columnWind.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnGen = new ColumnDefinition();
columnGen.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnError = new ColumnDefinition();
columnError.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnRatedPower = new ColumnDefinition();
columnGen.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnRatedWind = new ColumnDefinition();
columnRatedWind.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnLocation = new ColumnDefinition();
columnLocation.Width = new GridLength(350, GridUnitType.Auto);
ColumnDefinition columnGreenRate = new ColumnDefinition();
columnGreenRate.Width = new GridLength(250, GridUnitType.Auto);
ColumnDefinition columnGreenTotal = new ColumnDefinition();
columnGreenTotal.Width = new GridLength(250, GridUnitType.Auto);

docGrid.ColumnDefinitions.Add(columnName);
docGrid.ColumnDefinitions.Add(columnDate);
docGrid.ColumnDefinitions.Add(columnHour);
docGrid.ColumnDefinitions.Add(columnWind);
docGrid.ColumnDefinitions.Add(columnGen);
docGrid.ColumnDefinitions.Add(columnError);
docGrid.ColumnDefinitions.Add(columnRatedPower);
docGrid.ColumnDefinitions.Add(columnRatedWind);
docGrid.ColumnDefinitions.Add(columnLocation);
docGrid.ColumnDefinitions.Add(columnGreenRate);
docGrid.ColumnDefinitions.Add(columnGreenTotal);

TextBlock nameText = new TextBlock();
nameText.Text = weatherList[i].WindGenerator.Name;
nameText.HorizontalAlignment = HorizontalAlignment.Center;
nameText.Margin = thickness;
Grid.SetRow(nameText, i + 1);
Grid.SetColumn(nameText, 0);
TextBlock dateText = new TextBlock();
dateText.Text = weatherList[i].date.ToLongDateString();
dateText.HorizontalAlignment = HorizontalAlignment.Center;
dateText.Margin = thickness;
```

```
                Grid.SetRow(dateText, i + 1);
                Grid.SetColumn(dateText, 1);
                TextBlock hourText = new TextBlock();
                hourText.Text = weatherList[i].date.ToLongTimeString();
                hourText.HorizontalAlignment = HorizontalAlignment.Center;
                hourText.Margin = thickness;
                Grid.SetRow(hourText, i + 1);
                Grid.SetColumn(hourText, 2);
                TextBlock windText = new TextBlock();
                windText.Text = weatherList[i].windSpeed.ToString("0.00");
                windText.HorizontalAlignment = HorizontalAlignment.Center;
                windText.Margin = thickness;
                Grid.SetRow(windText, i + 1);
                Grid.SetColumn(windText, 3);
                TextBlock genText = new TextBlock();
                genText.Text = weatherList[i].WindGenerator.Power.ToString("0.00");
                genText.HorizontalAlignment = HorizontalAlignment.Center;
                genText.Margin = thickness;
                dailyTotal += weatherList[i].WindGenerator.Power;
                Grid.SetRow(genText, i + 1);
                Grid.SetColumn(genText, 4);
                TextBlock errorText = new TextBlock();
                errorText.Text = weatherList[i].WindGenerator.ErrorMessage;
                errorText.HorizontalAlignment = HorizontalAlignment.Center;
                errorText.Margin = thickness;
                Grid.SetRow(errorText, i + 1);
                Grid.SetColumn(errorText, 5);
                TextBlock ratedPowerText = new TextBlock();
                ratedPowerText.Text =
weatherList[i].WindGenerator.RatedPower.ToString();
                ratedPowerText.HorizontalAlignment = HorizontalAlignment.Center;
                ratedPowerText.Margin = thickness;
                Grid.SetRow(ratedPowerText, i + 1);
                Grid.SetColumn(ratedPowerText, 6);
                TextBlock ratedWindText = new TextBlock();
                ratedWindText.Text =
weatherList[i].WindGenerator.RatedWindSpeed.ToString();
                ratedWindText.HorizontalAlignment = HorizontalAlignment.Center;
                ratedWindText.Margin = thickness;
                Grid.SetRow(ratedWindText, i + 1);
                Grid.SetColumn(ratedWindText, 7);
                TextBlock locationText = new TextBlock();
                locationText.Text = weatherList[i].Area.Name == null ? "N/A" :
weatherList[i].Area.Name;
                locationText.HorizontalAlignment = HorizontalAlignment.Center;
                locationText.Margin = thickness;
                Grid.SetRow(locationText, i + 1);
                Grid.SetColumn(locationText, 8);
                TextBlock greenRate = new TextBlock();
                greenRate.Text = greenRateTB.Text;
                greenRate.HorizontalAlignment = HorizontalAlignment.Center;
                greenRate.Margin = thickness;
                Grid.SetRow(greenRate, i + 1);
                Grid.SetColumn(greenRate, 10);
                TextBlock greenTotal = new TextBlock();
                greenTotal.Text = (weatherList[i].WindGenerator.Power *
greenFareRate).ToString("0.00");
                greenTotal.HorizontalAlignment = HorizontalAlignment.Center;
                greenTotal.Margin = thickness;
                Grid.SetRow(greenTotal, i + 1);
                Grid.SetColumn(greenTotal, 11);
```

```
                docGrid.Children.Add(nameText);
                docGrid.Children.Add(dateText);
                docGrid.Children.Add(hourText);
                docGrid.Children.Add(windText);
                docGrid.Children.Add(genText);
                docGrid.Children.Add(errorText);
                docGrid.Children.Add(ratedPowerText);
                docGrid.Children.Add(ratedWindText);
                docGrid.Children.Add(locationText);
                docGrid.Children.Add(greenRate);
                docGrid.Children.Add(greenTotal);

                SolidColorBrush errorBrush = new SolidColorBrush(Colors.DarkRed);
                errorBrush.Opacity = 0.5;
                SolidColorBrush ratedWindBrush = new
SolidColorBrush(Colors.DarkGreen);
                ratedWindBrush.Opacity = 0.5;

                for (int j = 0; j <= 11; j++)
                {
                    Rectangle rectangle = new Rectangle();
                    rectangle.Stroke = Brushes.Black;
                    if (errorText.Text.Length > 0)
                        rectangle.Fill = errorBrush;
                    else if (double.Parse(windText.Text) >=
double.Parse(ratedWindText.Text))
                        rectangle.Fill = ratedWindBrush;
                    else
                        rectangle.Fill = Brushes.Transparent;
                    Grid.SetRow(rectangle, i + 1);
                    Grid.SetColumn(rectangle, j);
                    docGrid.Children.Add(rectangle);
                }
            }
            RowDefinition totalRow = new RowDefinition();
            ColumnDefinition columnTotal = new ColumnDefinition();
            ColumnDefinition columnTotalData = new ColumnDefinition();
            docGrid.RowDefinitions.Add(totalRow);
            docGrid.ColumnDefinitions.Add(columnTotal);
            docGrid.ColumnDefinitions.Add(columnTotalData);
            TextBlock totalData = new TextBlock();
            totalData.FontWeight = FontWeights.Bold;
            totalData.Text = totalPower.ToString("0.00");
            totalData.HorizontalAlignment = HorizontalAlignment.Center;
            totalData.Margin = thickness;
            Grid.SetRow(totalData, weatherList.Count + 1);
            Grid.SetColumn(totalData, 9);
            docGrid.Children.Add(totalData);

            RowDefinition totalGreenRow = new RowDefinition();
            ColumnDefinition columnTotalGreenData = new ColumnDefinition();
            docGrid.RowDefinitions.Add(totalGreenRow);
            docGrid.ColumnDefinitions.Add(columnTotalGreenData);
            TextBlock totalGreenData = new TextBlock();
            totalGreenData.FontWeight = FontWeights.Bold;
            totalGreenData.Text = greenFareLbl.Content.ToString();
            totalGreenData.HorizontalAlignment = HorizontalAlignment.Center;
            totalGreenData.Margin = thickness;
            Grid.SetRow(totalGreenData, weatherList.Count + 1);
            Grid.SetColumn(totalGreenData, 11);
            docGrid.Children.Add(totalGreenData);
```

```
            return docGrid;
        }

        private void CompareBtn_Click(object sender, RoutedEventArgs e)
        {
            if (!main.comparisonOn)
            {
                main.firstWeatherToCompare.Clear();
                main.comparisonOn = true;
                main.firstWeatherToCompare = main.weatherListHourly.Select(x =>
x.Copy()).ToList();
                MessageBox.Show("Choose another Wind Generator to compare and click
Calculate Power Button\n" +
                    " You can choose another location and Date Range, but it is
preferably not to change this data");
                this.Close();
            }
            else //if (main.comparisonOn)
            {
                main.secondWeatherToCompare.Clear();
                main.secondWeatherToCompare = main.weatherListHourly.Select(x =>
x.Copy()).ToList();
                ComparisonWindow comparisonWindow = new ComparisonWindow();
                comparisonWindow.Show();
            }
        }

        private void GreenRateTB_TextChanged(object sender, TextChangedEventArgs e)
        {
            GetGreenFareRate();
            greenFareLbl.Content = (modifiableTotalPower *
greenFareRate).ToString("0.00") + " €";
        }
    }
}
```

Додаток 3

Програмний агент управління та моніторингу вітроенергетичної установки

Опис програмного модулю

УКР.НТУУ“КПІ”.ТМ51106_19Б 13-1

Аркушів 6

2019

АНОТАЦІЯ

Розроблений програмний агент моніторингу та управління вітроенергетичної установки має функції моделювання процесу вироблення вітрової енергії вітроегенератором впродовж обраного проміжку часу, функції збереження звіту, процесу моделювання, можливість переглядати збережений звіт, функції порівняння як технічних характеристик вітряків, так і роботи вітярків.

Користувачами можуть бути люди, які мають девайс з доступом до мережі інтернет.

ЗМІСТ

1. ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

При створенні програмного забезпечення були використані такі засоби реалізації:

— Середовище розробки Microsoft Visual Studio, адже має найбільше функціоналу серед конкурентів;

— Мова програмування C# для написання логіки додатку;

— Фреймворк .NET для організації архітектури додатку;

— Шаблон проектування архітектури додатку Model-View-ViewModel для поділу моделі і її представлення, що необхідно для їх зміни окремо один від одного;

— Система для побудови клієнтських додатків Windows Presentation Foundation;

— Технологія Entity Framework для доступу до даних бази даних;

— Прикладний програмний інтерфейс Bing Maps API для додання роботи з мапою до проекту;

— Прикладний програмний інтерфейс DarkSky API для знаходження швидкості вітру за вказаний період часу;

— Git для версіювання розробленої системи;

— Пакет WPF Toolkit Data Visualization для виведення даних у графічному представленні;

— Мова розмітки XAML для організації інтерфейсу;

— Реляційну базу даних Microsoft SQL, вона є безкоштовную; має легкий та зрозумілий веб-інтерфейс та легко інтегрується з середовищем розробки Visual Studio;

1.1. Опис логічної структури

Для реалізації задачі моніторингу та управління вітроенергетичною установкою у складі мультиагентної системи був розроблений клієнтський додаток, підключений до локальної бази даних, що зберігає інформацію про вітряки.

Інтерфейс користувача — виконано з використанням таких технологій: XAML мова розмітки, C# — для написання логіки агента, WPF Data Toolkit Visualization, що надає інструментарій для візуалізації інформації у графічному вигляді.

1.2. Вхідні та вихідні дані

Вхідними даними для системи є дані по швидкості вітру за обраний користувачем період, дані користувача, що характеризують досліджувану місцевість, та вибраний вітрогенератор.

Вихідними даними є звіт з моделювання роботи вітроенергетичної установки, що несе у собі інформацію про вироблену вітряком енергію, швидкість вітру, стан вітряка, сумарну вироблену енергію за день, географічну точку де вітряк працює, номінальну потужність вітряка та номінальну швидкість вітру.

2. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано на персональному комп’ютері, який працює на базі процесору x64 Intel Core i7 (8th Gen) та має 8 Гб оперативної пам’яті. Розроблене програмне забезпечення працює на комп’ютерах з операційною системою Windows, що дозволяє запускати його на комп’ютерах будь-якої потужності.